MAXIMIZING CROSSTALK-INDUCED SLOWDOWN DURING PATH DELAY TEST

A Thesis

by

DIBAKAR GOPE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2011

Major Subject: Computer Engineering

Maximizing Crosstalk-Induced Slowdown During Path Delay Test

MAXIMIZING CROSSTALK-INDUCED SLOWDOWN DURING PATH DELAY TEST

A Thesis

by

DIBAKAR GOPE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Co-Chairs of Committee,	Duncan Henry M. Walker
	Jiang Hu
Committee Members,	Gwan S. Choi
	Jose Silva-Martinez
Head of Department,	Costas N. Georghiades

August 2011

Major Subject: Computer Engineering

ABSTRACT

Maximizing Crosstalk-Induced Slowdown During Path Delay Test.

(August 2011)

Dibakar Gope, B.E., Birla Institute of Technology and Science

Co-Chairs of Advisory Committee: Dr. Duncan Henry M. Walker
Dr. Jiang Hu

Capacitive crosstalk between adjacent signal wires in integrated circuits may lead to noise or a speedup or slowdown in signal transitions. These in turn may lead to circuit failure or reduced operating speed. This thesis focuses on generating test patterns to induce crosstalk-induced signal delays, in order to determine whether the circuit can still meet its timing specification. A timing-driven test generator is developed to sensitize multiple aligned aggressors coupled to a delay-sensitive victim path to detect the combination of a delay spot defect and crosstalk-induced slowdown. The framework uses parasitic capacitance information, timing windows and crosstalk-induced delay estimates to screen out unaligned or ineffective aggressors coupled to a victim path, speeding up crosstalk pattern generation. In order to induce maximum crosstalk slowdown along a path, aggressors are prioritized based on their potential delay increase and timing alignment. The test generation engine introduces the concept of alignment-driven path sensitization to generate paths from inputs to coupled aggressor nets that meet timing alignment and direction requirements. By using path delay information

obtained from circuit preprocessing, preferred paths can be chosen during aggressor path propagation processes. As the test generator sensitizes aggressors in the presence of victim path necessary assignments, the search space is effectively reduced for aggressor path generation. This helps in reducing the test generation time for aligned aggressors. In addition, two new crosstalk-driven dynamic test compaction algorithms are developed to control the increase in test pattern count. The proposed test generation algorithm is applied to ISCAS85 and ISCAS89 benchmark circuits. SPICE simulation results demonstrate the ability of the alignment-driven test generator to increase crosstalk-induced delays along victim paths.

DEDICATION

To my parents and family

ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to my advisor, Dr. Duncan M. (Hank) Walker, for his guidance and continuous support throughout the course of my thesis work. I would also like to thank him for guiding me with such dedication and consideration and never failing to pay attention to any details of my work. His technical insight, his novel ideas and his encouragement are all essential to this work. This thesis would never have been accomplished without his technical and editorial advice.

I would like to extend my gratefulness to the members of my advisory committee, Dr. Jiang Hu, Dr. Gwan S. Choi and Dr. Jose Silva-Martinez, for their guidance in my research. Thanks to my colleague Shayak Lahiri for his help in several software issues.

I would also like to thank the staff in the Department of Electrical Engineering for making my academic life at Texas A&M University a great experience. A special thanks to my friends, Reeshav Kumar, Akshay Godbole, Ayush Garg and Anurag Singla, for their support and encouragement which made this work possible. Finally, I am thankful to my parents for their love, encouragement and confidence in my abilities.

My research was funded in part by Semiconductor Research Corporation (SRC) and by National Science Foundation (NSF). I thank them for their financial support.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

1. INTRODUCTION

With continuous scaling of process technology in the very deep sub-micron (DSM) regime, the capacitive coupling between adjacent interconnect wires continues to increase and now dominates total interconnect capacitance. This leads to signal crosstalk noise. Interconnect delays are increasingly affected by signal crosstalk, leading to timing violations, reduced timing margin and signal glitches. Therefore, signal crosstalk noise must be considered in timing closure and manufacturing test. Capacitive crosstalk noise results from parasitic coupling between adjacent signal nets and is most seen in nets that have weaker drivers than adjacent nets [1].

Crosstalk faults can be categorized into two types: crosstalk-induced glitches and crosstalk-induced delays. A crosstalk-induced glitch [2] occurs when a victim line is intended to be in a stable state, but is found to have an unwanted noise pulse due to the transitions on one or more neighboring nets. Depending on their amplitude and width, these pulses can have an important impact on circuit performance [3]. A crosstalk-induced delay [4] is produced when both the affecting and victim lines have simultaneous or near simultaneous transitions. If the affecting net switches in the same direction as the victim net, it reduces the transition time of the victim. We refer to this phenomenon as crosstalk speedup. However, if the affecting and victim lines switch in

---

This thesis follows the style of *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*

the opposite direction, the victim line will experience an increase in delay, which is termed as crosstalk slowdown. In most circuits, crosstalk-induced delay, particularly slowdown delay, leads to the chip failure more so than the crosstalk-induced glitch [4] [5]. In current trends in integrated circuit design, it is impossible to eliminate errors caused by crosstalk noise because of stringent area and performance requirements. These crosstalk noises could be eliminated by resizing drivers, shielding interconnect techniques, rerouting signals and repeater insertion techniques. However, redesign may be very expensive in terms of design effort and its impact on a product's schedule. Moreover, due to the random nature of process variations, careful design and validation techniques cannot ensure all manufactured parts to be free of error-causing crosstalk effects. Thus testing for severe crosstalk noise effects is essential to guarantee the correct functionality of fabricated chips.

The need to magnify the impact of these crosstalk effects becomes increasingly important to reduce the probability of test escape of the delay-sensitive paths. Normal functional patterns cannot effectively maximize the crosstalk-induced delay effects along timing-critical paths. In addition to test these paths, these patterns need to model other functional use conditions in the remaining circuit to effectively detect other hard-to-detect logical defects. As a result, generating such efficient functional patterns that can maximize crosstalk-induced delays is a challenging task and can be prohibitively expensive. New automatic test pattern generation (ATPG) techniques are required to

maximize the coupling effects on critical paths while still ensuring high fault coverage and low pattern count.

As explained above, switching activity in capacitive-coupled nets can speed up or slow down the victim path if the nets involved in coupling have simultaneous or near simultaneous transitions. If the transitions at the affecting and victim lines occur at significantly different times (more than one gate delay), then there is no significant delay impact [4]. Moreover, in DSM circuits, physical synthesis avoids long parallel runs of signal nets, to minimize the noise from any one coupling capacitor. Because of the logical constraints and different timing windows of the aggressor sites, it is quite improbable for a single delay test pattern to excite a large number of aggressors on a single victim net. Significant crosstalk delay increases can only occur due to multiple aggressors coupling to multiple victim nets along a victim path.

Prior work on crosstalk ATPG does not consider the timing alignment of aggressor-victim nets and the impact of multiple simultaneous aggressor nets on a single critical path. As a result, the delay of the tested paths may be less than the worst case, leading to a test escape. New test pattern generation algorithms must focus on sensitizing a maximal subset of timing-aligned aggressors along the victim path under test.

The key contributions of this thesis are:

1. Timing-oriented test generation to target multiple aggressors along a victim path, so as to maximize the crosstalk delay.

2. Alignment-driven path sensitization to generate a path from primary inputs (PIs) to the coupled aggressor net that meets the required timing alignment and direction.
3. Two crosstalk-driven dynamic compaction algorithms to control the number of test patterns when incorporating crosstalk.

2. RELATED PRIOR WORK

Most of the prior work on testing for crosstalk has focused on logic faults caused by crosstalk induced glitches [1] [6] [7] [8] [9] and related test pattern generation techniques. Testing for crosstalk-induced delays has recently received more attention [10] [11]. Several fault models and test generation techniques have been proposed to take into account crosstalk-induced delay. The common objective of all these techniques is to find the most effective set of patterns causing maximum crosstalk-induced delay along timing-critical paths. Since the pattern generation for crosstalk induced delay faults requires timing information, reducing the high complexity of the ATPG process is a major issue for prior test-generation methods.

The timing-oriented backtrace procedure proposed in [4] and [12] considered timing alignment of the aggressor with the victim net in pattern generation. However, this approach did not take into account the possible influence of multiple aggressors for a given victim net or the effect of multiple victim nets on a single critical path. Essentially the coupling capacitance to overall net capacitance ratio considered was large enough that by propagating on the longest path and sensitizing the worst case, the coupling slowdown would be detected. This is not feasible in modern DSM circuits. Focused on all aggressor lines of a victim line, the authors in [13] proposed a solution that combines an integer-linear program with the traditional stuck-at fault ATPG. These two methods could not activate the worst case crosstalk-induced delay, since they

consider testing of the crosstalk effect on a single victim line, similar to transition fault delay testing, without considering accumulated delay defects or effects on a path.

The work presented in [14] and [15] employs an algorithm based on boolean satisfiability (SAT) wrapped by a branch-and-bound algorithm to find the subset of aggressors exciting maximal crosstalk noise on a victim line. As a result, false noise can be reduced in order to provide a more accurate static timing analysis. Since no previously generated test is given as a constraint, this approach is guaranteed to calculate the subset of aggressors providing maximal crosstalk noise but without taking test generation into account. The drawback of this approach is the long run time.

The authors in [16] proposed a test generation method for critical paths considering single aggressor crosstalk effect with due consideration to the timing alignment and direction. This method has similar CPU efficiency to that of [17] and [18]. However, they did not take into account the possible impact of multiple aligned aggressors along a victim path. In addition, the backtrace procedure does exhaustive search for aggressor pattern generation and so this methodology suffers from computational complexity.

The ATPG technique in [19] applied boolean constraints and modified PODEM algorithm to construct a heuristic solution that excited multiple aggressors on a target path. In [20] the authors presented a constrained path delay fault (CPDF) model as a combination of a timing-critical path and a set of crosstalk noise sources interacting with the path. However, the technique was computational intensive because it was based on

genetic algorithm and did not efficiently handle timing information. A timed ATPG method is proposed in [21] to generate critical paths and corresponding input vectors to sensitize these paths under crosstalk effects. This approach incorporated special timing processing techniques into ATPG algorithms and employed expensive circuit-level timing simulation. These three methods are not scalable to industrial circuits.

In [22] the authors used timing-driven boolean logic to characterize signal transitions in a time interval. Moreover, they employed boolean satisfiability (SAT) technique to check the correlations between aggressor and victim transitions. The authors in [10] incorporated the sensitization of a maximal set of potential coupled aggressors in a transition fault framework. This has the advantage of reusing the existing transition fault infrastructure, but the disadvantage of not being able to determine timing alignment. These two methods can find the patterns efficiently by means of ignoring the timing of aggressors, but they could not guarantee the timing requirements for activation of the targeted crosstalk effects.

To generate deterministic test patterns for crosstalk-induced delay faults, timing information cannot be ignored. However, including timing information into an ATPG engine will significantly increase the complexity of the ATPG algorithm. Considering the timing of the aggressors is the main obstacle for efficient test generation.

3. CROSSTALK-INDUCED DELAY MODELING

Crosstalk is caused by parasitic coupling between adjacent signal nets that include inductive and capacitive effects. On-chip inductance becomes significant at high frequency in certain global signal lines, such as VDD and ground buses. However, capacitive coupling tends to dominate for signal interconnects. So it is still possible to accurately model crosstalk-induced delay effects without considering any impact from inductance.

Signal crosstalk between a victim net and its neighboring aggressor nets may either speed up or slow down the victim path depending on the transition direction, transition arrival time overlap and coupling capacitance between the victim and aggressor nets [1] [23]. This work will focus on signal slowdown.

Coupling models proposed in the literature can be broadly classified into two categories: charge sharing based coupling models and simulation based coupling models. Transitions on aggressors change the effective capacitance (C_{eff}) seen by the victim net driving gate and thus change the signal transition delay. In charge sharing based coupling models, crosstalk is modeled by scaling the physical coupling capacitance (CC) with a Miller Coupling Factor *(MCF)* to obtain the effective coupling capacitance value.

$$C_{eff} = MCF \; . \; CC$$

If the aggressor transition occurs at a significantly different time than the victim net (more than one gate delay), then there is no significant delay impact [4]. If the aggressor transition overlaps with the victim transition and is in the same (helper) direction, then C_{eff} is reduced and the victim speeds up. If the aggressor transition is in the opposite (aggressor) direction, then C_{eff} is increased and the victim slows down. In addition, crosstalk delay noise depends on other factors such as slew rate [1] [24] and drive strength of victim-aggressor pair [1] [25].

For aggressor and victim switching in opposite directions, the MCF factor can take values from 1 to 3 [26]. If the coupled aggressor net has a much faster transition time than the on-path victim net, then an MCF greater than 2 can result. A probabilistic linear model is proposed in [27] to estimate the MCF. Given the minimum relative signal arrival times estimated for a victim-aggression pair, the authors in [28] can determine the corresponding MCF using a regression based model. The dependence of delay noise on the alignment can be computed by using circuit simulations [29] or derived analytically using curve-fitting techniques [30].

Charge-sharing based coupling models are used chiefly in the early stages of design flow because of their efficiency. In this work, we focus on early stages of the design flow and therefore use a charge sharing based coupling model. Our algorithm can be extended to an accurate crosstalk delay model [31], but details of the extension are beyond the scope of this work.

The following delay equation is used to estimate the crosstalk-induced delay increase caused by *i*-th aggressor switching at the same time, but in the opposite direction, as the coupled victim net:

$$\Delta Delay_{crosstalk} = \left(\frac{C_{c_i}}{C_l + \sum_{i=1}^{n} C_{c_i}}\right). Delay_{driver-stage}$$

where $\Delta Delay_{crosstalk}$ is the crosstalk-induced delay increase caused by the *i*-th aggressor; C_{c_i} denotes the coupling capacitance between the *i*-th aggressor and the victim net, C_l is the line capacitance from the victim net to ground, n is the number of aggressors, and $Delay_{driver-stage}$ is the nominal stage delay of the victim net, assuming no transitions on the coupled nets. The denominator of the equation is the nominal value of C_{eff}. This equation approximates the delay as linear in the change in C_{eff}. Further equal aggressor and victim slew rates are assumed with completely overlapping transitions, so the MCF is 2. In practice, this is the maximum potential delay increase. We assume linear superposition of aggressor noise, so the aggressor noise coupled to a victim net can be aggregated linearly. Thus nonlinearity of parasitic and Miller effects are ignored in this work. Further we do not consider the impact of aggressors on each other to compute the potential delay increase on the victim net.

Coupling noise is a significant issue for relatively long signal nets. These nets tend to be routed through multiple metal layers. We have found from the RC extraction of the

ISCAS circuits that typical long nets are capacitive-coupled to 40–50 other signal nets. However, out of those 40–50 neighboring nets, only 4-5 make up 80%–90% of the total coupling capacitance value. As a result, if we can generate aligned aggressor transitions on 4–5 significant coupled nets, we can come close to producing the worst case crosstalk delay on the victim net without creating implausible ATPG requirements.

The afore-mentioned delay model considers the nominal stage delay of the victim net $Delay_{driver-stage}$ in the crosstalk delay increase computation. However, a change in input-signal slope caused by crosstalk can impact the nominal stage delay at its receiving gates. In addition, the impact of noise on a signal line may affect its receiver gates differently because of varying logical thresholds of the receivers. These effects are not considered in this work.

4. PROPOSED TEST GENERATION FOR CROSSTALK-INDUCED DELAY

Our proposed crosstalk-induced delay test generation procedure consists of three major steps: (1) K longest path per gate (KLPG) [32] test generation for a delay-sensitive path; (2) sorting and pruning aggressors along each victim path, based on logic constraints, timing alignment, and their potential delay increase; and (3) path generation from PIs to the aggressor nets that meets the timing alignment and transition direction.

For a set of potential aggressors coupled to a delay-sensitive victim path, the objective of this proposed ATPG is to generate a test vector that can excite maximal number of aligned aggressors while also sensitizing the victim path.

4.1 KLPG Test Generation

In this work, KLPG test generation [32] is used to generate the longest path through every line in the circuit under robustness constraints. The target line or fault site is assumed to have a spot delay defect. Figure 1 shows the basic flow of the KLPG path generation algorithm.

The search space for each fault site is the fan-in and fan-out paths of the target line. Paths outside the search space can provide side input constraints for gates on the path.

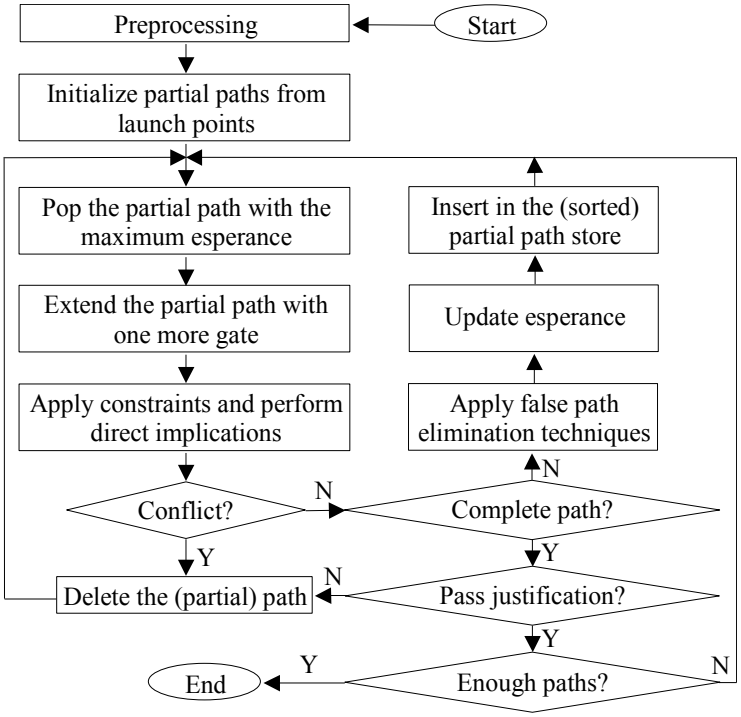

Figure 1 KLPG path generation algorithm

In the path generation phase, a path store is used to store partial paths, which are paths originating from a PI but have not reached a PO. Every partial path has a value called esperance [33], which is the sum of the length of the partial path and the min-max path delay from its last node to a PO, without considering any logic constraint. In other words, the maximum esperance is the upper bound on the length of a partial path that grows to be a complete path, and the minimum esperance is the lower bound. As shown in Figure 1, in each iteration of path generation, the partial path with the largest max esperance is popped from the sorted path store and extended by adding one fan-out gate

with largest max esperance. If the last gate of the partial path has multiple fan-outs, the path will split, leaving the alternate choices in the path store. Depending on the sensitization criterion, such as robust or non-robust sensitization, constraints to propagate the transition on the added gate are applied. Then direct implications [32] are performed to identify local conflicts. A direct implication on a gate is one where the input or output value of that gate can be determined from other values assigned to that gate. Previous research [32] [33] found that direct implications can eliminate most false paths. If a partial path reaches a PO, it becomes a complete path. Then a PODEM-based final justification [32] is performed to find a vector pair that sensitizes this path. Since the longest path through one line may be the longest path through other lines, a new complete path must be checked to see if it has already been generated before. The test generation repeats until the K longest testable paths (both rising and falling transitions) through each line are generated or the path store is exhausted.

When a path is generated and passes final justification, a set of necessary assignments (values assigned to lines) are identified that are necessary to sensitize and propagate the fault along the path. Assignments generated during final justification are not saved, since they may not be necessary. Assignments generated during victim path generation are used to screen out the coupled aggressors that have a helper transition or constant values set from the victim path necessary assignments (NAs).

Crosstalk-induced delay increases are relatively small. They are only of concern if the delay defect on the path under test is large enough that the path is almost failing, but

not so large that the path fails regardless of crosstalk. Thus aggressors are aligned assuming that the target delay defect is equal to the path timing slack. This shifts the nominal transition times downstream from the defect site, as shown in Figure 2.

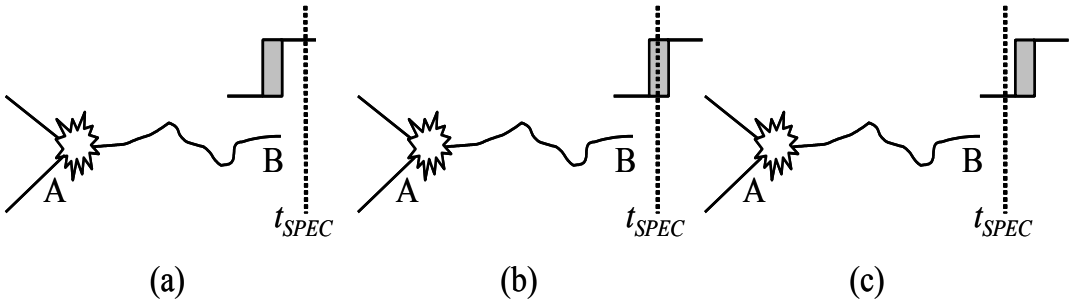

Figure 2 (a) Path never fails, no crosstalk impact; (b) crosstalk can cause delay fault; (c) path always fails, no crosstalk impact

Maximizing the victim path delay increase is a form of the maximum cover problem. The cost of a near-optimal solution for this problem does not make sense given our timing model approximations. We instead use a greedy algorithm, targeting aggressors in decreasing order of potential delay increase. This works well when a small number of larger coupling capacitances dominate the potential delay increase.

4.2 Aggressor Pruning and Ranking

4.2.1 Delay Threshold Pruning

Aggressors that do not cause victim path delay increases are pruned away. Cadence SoC Encounter is used to extract the Standard Parasitic Exchange Format (SPEF) file for ISCAS85 and ISCAS89 circuits. SPEF stores the parasitic information of the nets used in the layout.

First, the NAs to sensitize the victim path may propagate to aggressors. These aggressors are discarded since there is no decision to be made. Second, many coupling capacitors are small, and cause insignificant victim path delay increase. Aggressors are retained only if their potential delay increase metric is above a specified threshold:

$$\Delta Delay_{crosstalk} \geq ThresholdDelayIncrease$$

where $\Delta Delay_{crosstalk}$ is the crosstalk-induced delay increase caused by an aggressor, defined in Section 3. The threshold is set by analyzing victim path delay increase vs. threshold to determine an appropriate delay vs. cost trade-off. We term this pruning as delay threshold pruning.

4.2.2 Logical and Alignment-Based Pruning

Along a path there are multiple logic stages, each one having many coupled nets. The worst path delay due to crosstalk would be for all coupled nets to have aggressor

transitions aligned with the victim transition on the tested path. However, this case is not possible due to logic and timing conflicts:

1. Some coupled nets have NAs for the testing of the path that preclude an aggressor transition, or in fact mandate a helper transition.
2. Some coupled nets can only have transitions that do not align with the tested path transition. For example, if a net on a short path is coupled near the end of a long path.
3. Some aggressor transitions have logic constraints that conflict with other aggressor transitions. For example, one transition may have an NA that precludes another transition.
4. Some aggressor transitions have timing alignment constraints that conflict with other aggressor alignment requirements. For example, if a net couples to two different logic stages on the tested path, only one of the couplings can be aligned.

Logical and alignment-based pruning is used to screen out cases #1 and #2.In our research, we combine cases #3 and #4 together by modifying our existing KLPG path generator to generate paths from the PIs or PPIs to the coupled net location with the necessary parity (transition direction) and timing alignment.

Direct implications are applied on remaining aggressors obtained from delay threshold pruning. The aggressor net is assigned a transition opposite to that of the victim net and direct implication is used to propagate values. During direct implications,

if a conflict is found with the NAs to sensitize the victim path, the aggressor is not considered for further alignment checking and sensitization. We term this pruning as logical pruning. This direct implication trims off the false-aggressor candidates in the initial phase of aggressor ranking, else an aggressor may be selected that will fail during its path sensitization, wasting ATPG work.

Next, a static timing analysis (STA) engine computes the earliest and latest possible rising/falling transition timing windows on the input and output lines of each gate in the circuit, using the victim path NAs. Assuming the transition at PIs at time zero, the engine traverses the circuit starting from PIs in a breadth-first manner to compute timing windows for each line. If the aggressor and victim net timing windows do not overlap, the aggressor is pruned. The victim path NAs significantly narrow aggressor timing windows and ease the identification of more accurate time-aligned aggressor. Since prior work [17] [34] did not use the victim path NAs to compute the earliest and latest possible rising/falling transition timing windows like traditional STA calculations, timing windows generated by those methods are very pessimistic and have very wide ranges. This may lead to missing the real aggressor lines for the target victim path.

This three-step pruning is performed on each aggressor coupled to a victim net and for the aggressors to the other victim nets along the same target path.

After pruning, aggressors are inserted into the potential aggressor list for the victim path in decreasing order of *coupling effectiveness*. The *coupling effectiveness* metric is defined as:

$$TimingWindow = T_{Aggr}(Latest) - T_{Aggr}(Earliest)$$

$$LeftWindow = T_{Victim} - T_{Aggr}(Earliest)$$

$$RightWindow = T_{Aggr}(Latest) - T_{Victim}$$

$$WindowDiff = RightWindow - LeftWindow$$

$$CouplingEff = \frac{C1 \cdot \Delta Delay_{crosstalk}}{C2 \cdot TimingWindow + C3 \cdot WindowDiff}$$

where $CouplingEff$ is the *coupling effectiveness* of the aggressor, T_{Agg} and T_{Victim} denote the aggressor and victim transition times and *C1*, *C2*, *C3* are user-defined constants. Aggressors that have higher potential delay increase, more symmetric overlap of aggressor and victim timing windows, and a smaller timing window, will be ranked higher in the potential aggressor list. The overall aggressor pruning flow is shown in Figure 3.

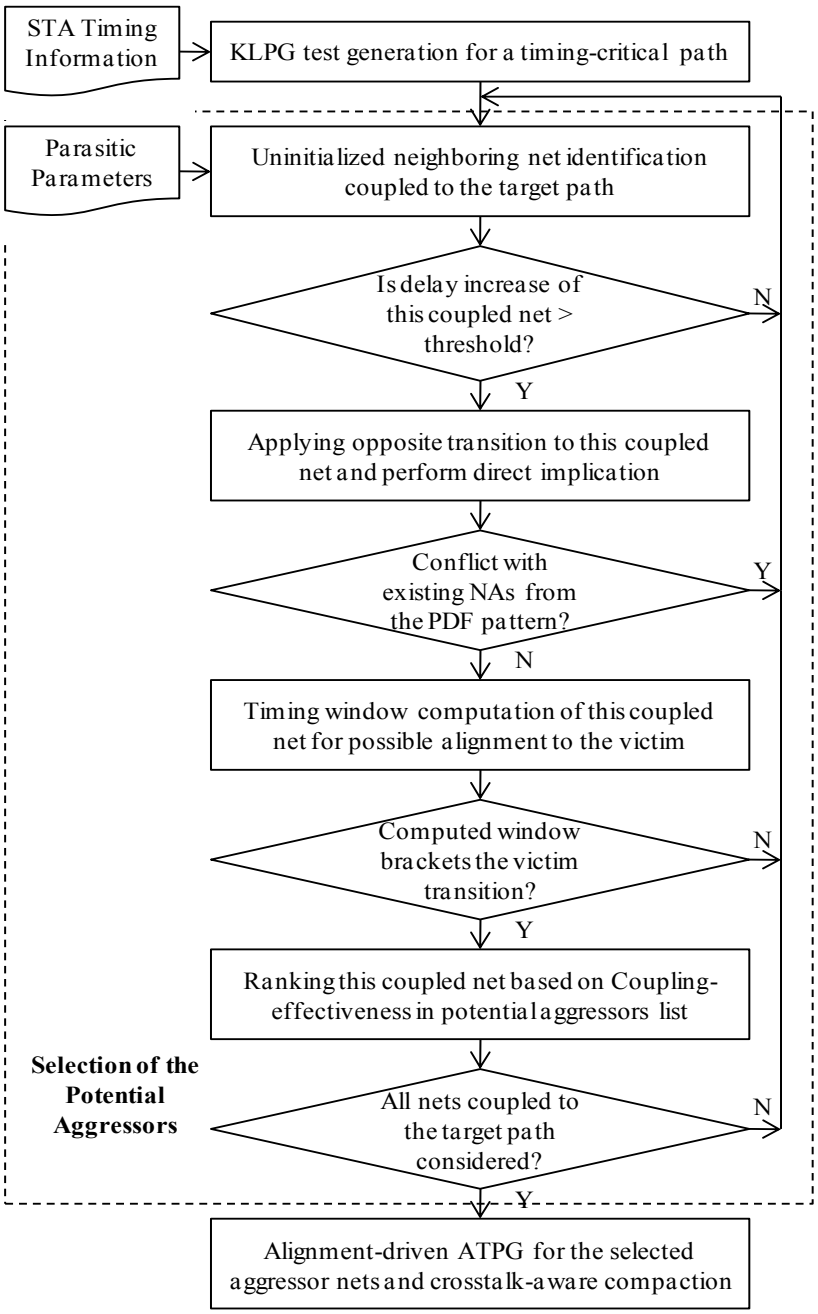

Figure 3 Aggressor pruning algorithm

5. TIMING-ORIENTED ATPG

After sorting the potential aggressors for a victim path, the aggressor with maximum *coupling effectiveness* is considered for sensitization in presence of victim path NAs. The goal is to find a propagation path from the PIs to the aggressor that has the best timing alignment. In practice, alignment is probabilistic, depending on process variation, supply noise, and other unmodeled effects. Activity in one part of the circuit can throw off the alignment in another part of the circuit. Since we are concerned with paths that are too slow, the alignment requirement can be indirectly accounted for by using min-max gate delays in the *coupling effectiveness* ranking. In our work, we will use nominal circuit delays during the search for the path to the aggressor that achieves the best alignment, ignoring any lack of correlation due to noise or process variation. Each aggressor shifts the timing alignment of later nets on the victim path. This can be handled by updating transition times along the victim path, but the shift in alignment is small enough that this is not considered.

5.1 Path Store

The KLPG engine was modified to sensitize aligned aggressor transitions. In the path generation phase, a path store is used to store partial paths, which are paths originating from a PI but have not reached the aggressor of interest. The search space for each aggressor net, as shown in Figure 4, is the fan-in cone of the aggressor line. Paths outside the search space can provide side input constraints for gates on the path. Figure 5

shows an example. The partial path starts from primary input g_0, and ends at gate g_i. A set of partial paths are grown from PIs towards the aggressor net, with the goal of sensitizing a path to the aggressor and achieving the best timing alignment with the victim net. At the beginning, the path store attempts to generate $2n_{PI}$ partial paths, where n_{PI} is the number of primary inputs in the fan-in cone of the aggressor line. Partial paths are initialized as rising and falling transitions from all the PIs of the aggressor fan-in cone that do not already have NAs. When a partial path reaches the aggressor net, it becomes a complete path.

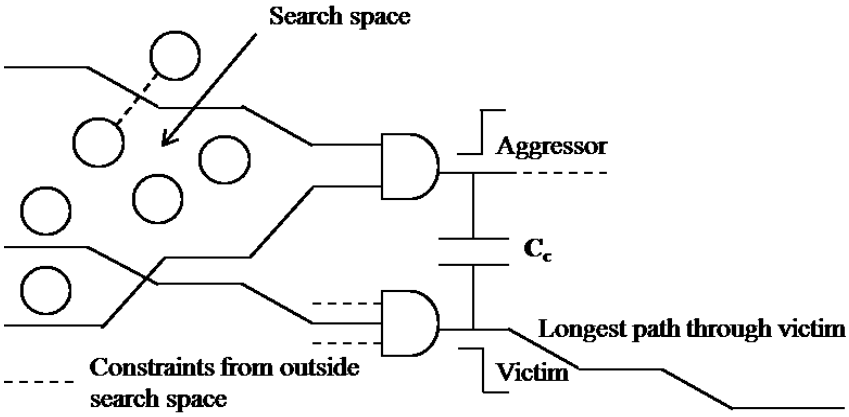

Figure 4 Aggressor path search space

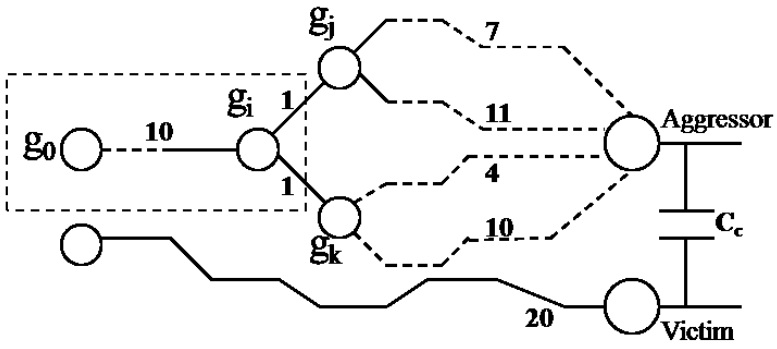

Figure 5 A partial path

The earliest and latest aggressor transition times are associated with each partial path. These are the sum of the length of the partial path and the min/max path delay from its last node to the target aggressor. The partial paths are sorted by their potential timing-alignment to the victim net. The timing alignment metric is calculated as:

$$TimingWindow = Delay_{Aggr}(Max) - Delay_{Aggr}(Min)$$

$$LeftWindow = T_{Victim} - Delay_{PartialPath} - Delay_{Aggr}(Min)$$

$$RightWindow = Delay_{PartialPath} + Delay_{Aggr}(Max) - T_{Victim}$$

$$WindowDiff = RightWindow - LeftWindow$$

$$TimingAlign = \frac{1}{C2 \cdot TimingWindow + C3 \cdot WindowDiff}$$

where *TimingAlign* is the timing alignment metric. $Delay_{Aggr}$ is the path delay from the last node of the partial path to the target aggressor. $Delay_{PartialPath}$ is the length of the partial path. The other variables are as described earlier. In Figure 5, suppose the length of the partial path $g_0...g_i$ is 10 and the min/max path delay from g_i to the aggressor is 5/12. The victim net transition timing is shown as 20. Assuming the value of $C2$ and $C3$ as 0.25 and 0.25 respectively, the potential timing-alignment of this partial path is 0.4.

5.2 Path Generation

In each iteration of path generation, the partial path with the largest timing alignment value is popped from the path store and extended by adding a fan-out gate that achieves the maximum alignment. If the last gate of the partial path has multiple fan-outs, the path will split, leaving the alternate choices in the path store. In order to target an aligned aggressor, the timing-driven ATPG always propagates on the fan-out tree whose minimum and maximum delays bracket the victim net transition. One challenge is that several fan-out trees may meet this requirement. The heuristic we use to make a selection is to choose the fan-out tree that most evenly brackets the required delay and has the smallest delay range. Intuitively, as a path is built from inputs to outputs, the minimum length increases and the maximum length decreases, as false paths are ruled out. For example, in Figure 6, the partial path $g_0...g_i$ is extended by adding gate g_j, because extending to g_j could potentially give the best possible aggressor alignment to the victim transition. After the partial path is extended ($g_0...g_ig_j$ in Figure 6), the constraints to propagate the transition on the added gate (g_j) are applied. Then direct

implications are used to check the compatibility of the new partial path NAs with the existing NAs from the victim path.

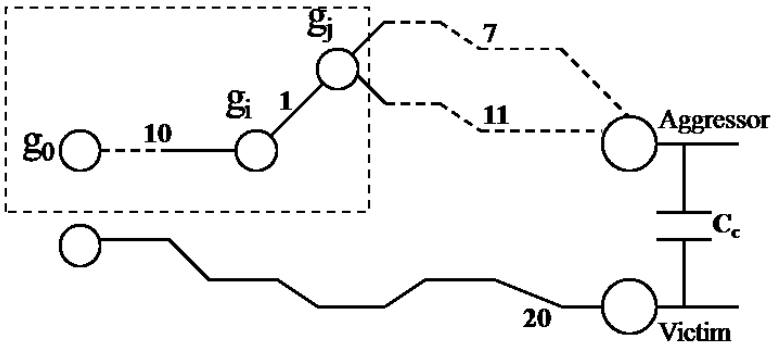

Figure 6 Extending a partial path

If a conflict happens during direct implications, the partial path is false. In other words, any path including this partial path is a false path. Therefore, the partial path is deleted from the path store so that the whole search space which contains this partial path is trimmed off. If a partial path reaches the target aggressor, it becomes a complete path. It also means the NAs from the aggressor path sensitization are compatible with the existing victim paths NAs. Then a PODEM-based final justification is performed on the combined sets of NAs to find a test pattern that simultaneously sensitizes the victim path and the aggressor net.

Once a partial path reaches the desired aggressor net, it is not further propagated to an observable point, because propagating the aggressor transition further may create

additional NAs that can be better used to activate an opposite transition at another aggressor net.

The NAs for the justification of this aggressor are retained when searching for later aggressor paths. The path generation procedure is repeated for all other aggressors in decreasing order of potential delay increase. The process is repeated for all victim paths. The path generation process is shown in Figure 7.

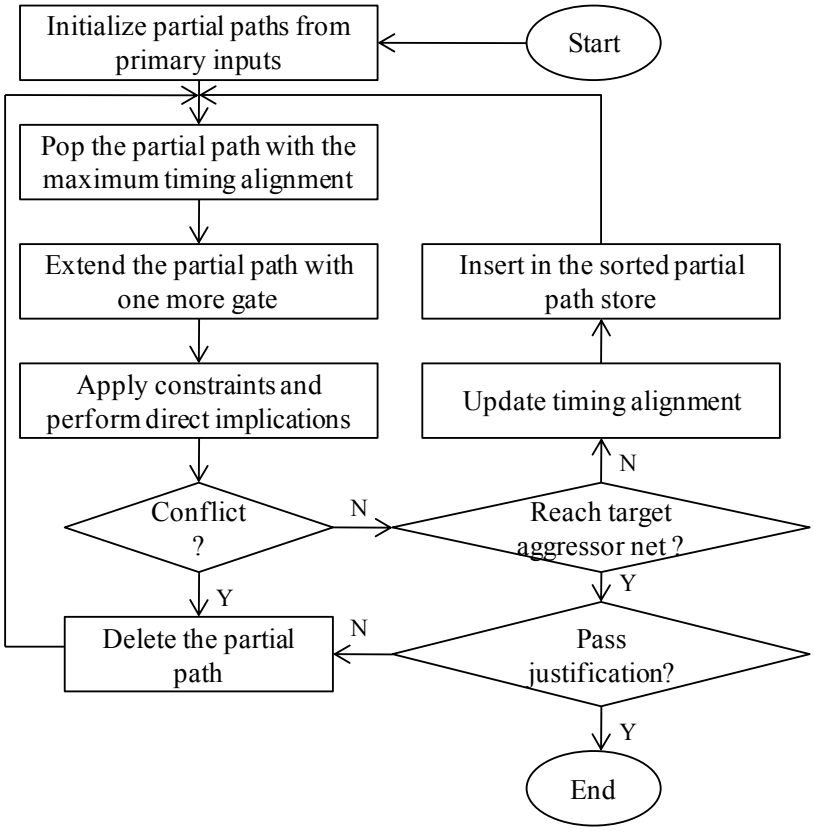

Figure 7 Path generation algorithm

Procedure *Timing-Oriented Aggressor Sensitization*() describes the alignment-driven aggressor path generation flow as follows:

Procedure *Timing-Oriented Aggressor Sensitization*()

1. Find the longest path through a target line. Justify the necessary assignments, but do not keep the primary input values.
2. Find the next aggressor coupling that would cause the largest path delay increase. If the potential delay increase due to this aggressor is less than the specified threshold, END.
3. Check whether this coupled net can have an aggressor transition on it. If not, go to step 2.
4. Check whether this coupled net can have timing alignment with the path net. If not, go to step 2. This can be easily checked by computing min-max delays of each net using breadth and depth first search.
5. Generate a path from PIs to the coupled net that meets the timing alignment and direction.
6. Justify the necessary assignments of the tested path and all coupling paths, but keep only the necessary assignments, as in step 1. If justification fails, discard the aggressor coupling. Go to step2.

6. CROSSTALK-AWARE DYNAMIC COMPACTION

Capacitive crosstalk has a relatively small impact on path delay compared to path length, supply noise or temperature. At one time, the optimal logic depth in microprocessors was thought to be 6 to 8 gates. However, logic depth is currently rising to meet low power requirements. If we assume a logic depth of 10, and a coupling capacitance of about 10% of the total net capacitance, then one aggressor transition can increase path delay by at most 1%. Prior work suggests at most a few percent delay increase due to crosstalk. So in order to cause substantial crosstalk-induced delay along the targeted victim path and subsequently to push that path towards delay test failure, the crosstalk pattern generation should attempt to excite maximal possible number of aggressors with required timing alignment and direction along a delay-sensitive path.

Once aggressors are sensitized using our timing-oriented ATPG, they are combined together into the test pattern of the victim path in decreasing order of their *coupling effectiveness*. That way the final compacted pattern will tend to activate as many high-impact aggressors as possible along a victim path to maximize the impact of crosstalk slowdown. This algorithm is greedy, so it may miss the worst possible crosstalk delay increase, both due to the order dependence, and stopping when the couplings are too small. However, in our experience, there are relatively few significant coupling capacitances and many insignificant ones, and a greedy algorithm will come

close to achieving the worst-case delay increase, particularly when considering the fact that timing alignment is uncertain due to intra-die process variation.

It is typically the case that many path tests can be dynamically-compacted into one test pattern [35]. This significantly reduces test pattern count over static compaction [36]. There are two approaches to using dynamic compaction when considering crosstalk.

6.1 Aggressor-First Dynamic Compaction

One approach to dynamic test compaction is to first compact the maximal number of aggressors into the test pattern for each victim path. The NAs of the victim path and aggressors sensitized so far are used to constrain the search for later (lower potential delay increase) aggressors, as shown in Figure 8. We term this aggressor-first dynamic compaction, since we first compact as many aggressors as possible per victim path, then compact these groups of victim and aggressors together into patterns.

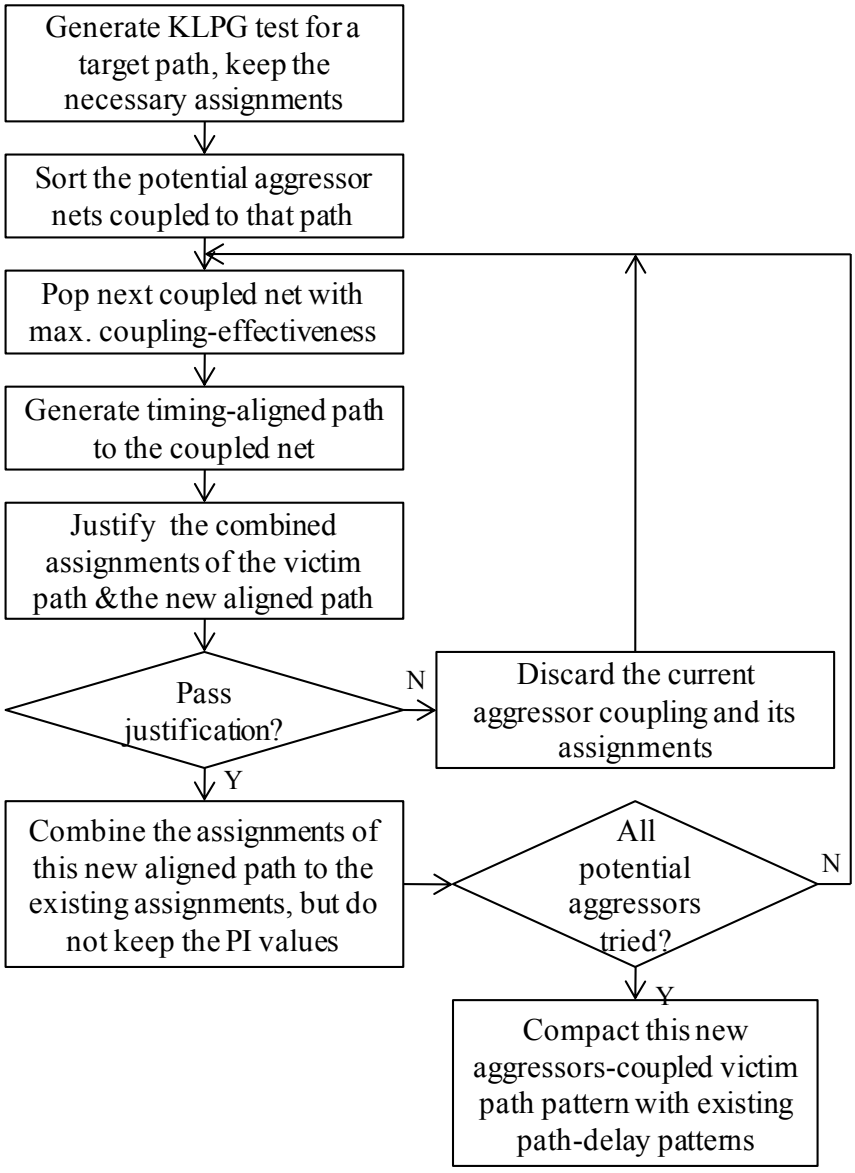

Figure 8 Aggressor-first dynamic compaction

Procedure *Aggressor-First_dc*() describes the crosstalk pattern generation flow with aggressor-first dynamic compaction. *POOL-Aggr* and *POOL-Victim* are the data

structures created to save the compacted aggressor pattern set for each victim path and the final compacted pattern pool respectively.

Procedure *Aggressor-First_dc*( )

1. Initialize the pattern pool *POOL-Victim* as empty.
2. Initialize the pattern pool *POOL-Aggr* as empty.
3. Use KLPG to generate a longest path *I* through a line, resulting in pattern *F*.*F* contains the NAs before justification.
4. Sort the potential aggressors coupled to that delay-sensitive victim path.
5. Pop the next potential aggressor with maximum *coupling effectiveness*. If the potential aggressor list for a victim path becomes empty, go to step 9.
6. Use timing-oriented ATPG to generate aligned aggressor transition.
7. Do final justification of the combined NAs from the victim path and the new aligned aggressor path. Do not keep the NAs from final justification.
8. If justification fails, destroy the NAs from the new aggressor path and go to step 5. Else Call procedure *Dyn_compact*(*F*, *POOL-Aggr*) and go to step 5.
9. For each and every pattern *P*in *POOL-Aggr*, call procedure *Dyn_compact(P*, *POOL-Victim)*. Go to step 2.
10. Do final justification for all patterns in *POOL-Aggr* one by one to generate the final vectors.

The dynamic compaction procedure *Dyn_compact*(*F*, *POOL)* [35] uses a greedy approach, in which each new pattern *F* is compacted with the first compatible pattern in

POOL. Patterns in *POOL* are sorted by non-increasing order of the number of necessary assignments in order to compact as many as possible paths into a pattern before it is written out. In contrary to static compaction, dynamic compaction algorithm checks the compatibility between necessary assignments, greatly expanding the compaction space without loss of fault coverage. Clearly the first pattern in any aggressor path-pool *POOL-Aggr* will sensitize the maximal number of time-aligned aggressors coupled to that victim path. In practice, the number of coupled nets that can be sensitized for a victim path using a single pattern is not large. As each aggressor is set, it adds more NAs that rule out other aggressors to sensitize.

6.2 Pattern-First Dynamic Compaction

The aggressor-first compaction procedure will maximize the crosstalk-induced delay increase on each victim path, but may cause an increase in the number of test patterns, compared to a test set that does not consider crosstalk. This pattern inflation can be avoided by first compacting victim paths and then sensitizing aggressors, which we term pattern-first compaction, as shown in Figure 9. The coupled nets to a victim path are sensitized in the presence of NAs from all the victim paths in a compacted pattern. The additional NAs in each pattern due to the victim paths will preclude sensitization of many aggressors. The same process will be repeated for the other compacted patterns in the set. Within a pattern, victim paths will be targeted in decreasing order of length.

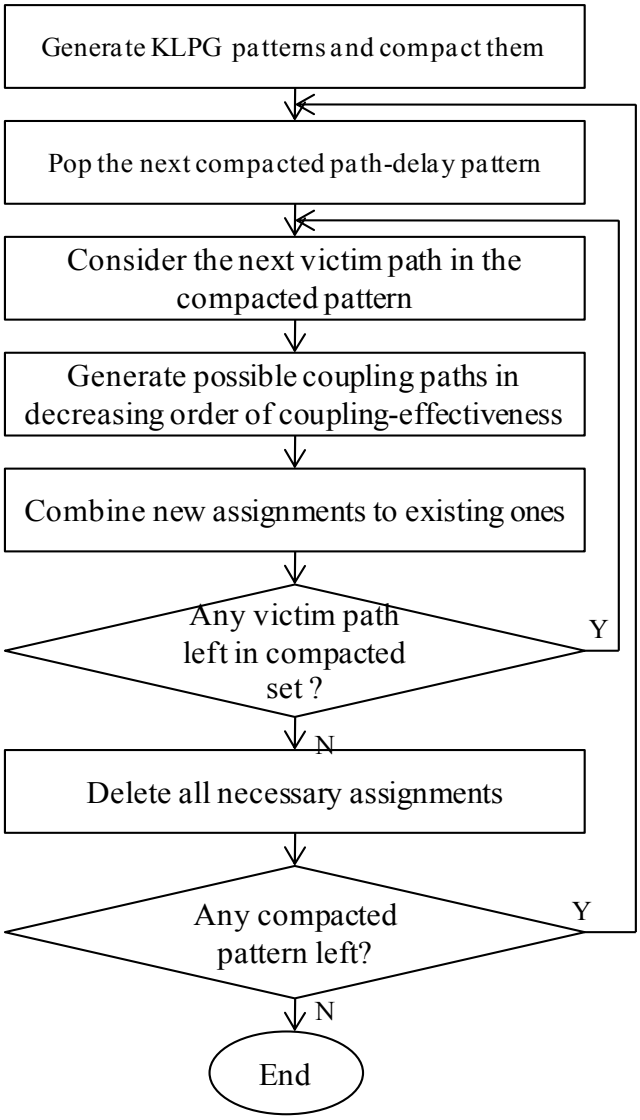

Figure 9 Pattern-first dynamic compaction

Procedure *Pattern-First_dc*() describes the crosstalk pattern generation flow with pattern-first dynamic compaction. *POOL* is the data structure created to save patterns.

Procedure *Pattern-First_dc*( )

1. Initialize the pattern pool *POOL* as empty.
2. Use KLPG to generate a longest path *I* through a line, resulting in pattern *F*. *F* contains all NAs before justification. If no more paths can be generated or we have enough paths, go to step 4. Otherwise go to step 3.
3. Call procedure *Dyn_compact*(*F, POOL*). Go to step 2.
4. Do final justification for all patterns in *POOL* one by one to generate the compacted victim path patterns.
5. Pop the next compacted pattern from *POOL*. If no more compacted pattern is left in *POOL*, procedure is finished.
6. Consider the next victim path in the compacted pattern. If no more victim path is left in compacted set, go to step 9.
7. Generate possible coupling paths in decreasing order of *coupling-effectiveness* in presence of NAs from all victim paths in compacted set.
8. Combine new NAs to existing ones. Do not keep NAs from any final justification of compacted victim pattern and new aligned aggressor paths. Go to step 6.
9. Delete all NAs. Go to step 5.

7. LOW-COST METRIC

We have proposed a realistic low cost fault coverage metric to detect the combination of a delay spot defect, process variation and crosstalk-induced slowdown. Our goal in using the low cost fault coverage metric is to reduce crosstalk-aware pattern count by dropping victim paths with large slack from crosstalk delay-induced pattern generation.

In many designs, there are a set of speed paths that determine the clock cycle time, and most fault sites have relatively short paths. The authors in [37] reported that the average longest path through each line is much shorter than the longest path length in ISCAS89 circuits. For example, for s38417, the longest path length is 41 gate delays, while the average length is 18.1.

In general, a precise physical model to reflect the real process and defect environment is not available. Even if available, it would be too costly to use during crosstalk pattern generation. In order to minimize test generation time, a simple model is desired. In this work, we use three criteria to set a detection probability threshold $P_{threshold}$. First, we assume the process variation is independent for each path and influences delay by increasing the required delay guard band. The percentage bound α covers the influence of inter-die and intra-die variation [38], power supply and substrate noise. Second, we consider that the spot delay defect size due to resistive short or open has a guard band. The defect size to exceed this guard band requires a bridge resistance

so small or open resistance so large that it nearly causes a transition fault. Third, the total crosstalk-induced delay from all the potential aggressors coupled to a victim path increases the required delay guard band. As the majority of the potential aggressors after pruning cannot be sensitized in the presence of victim path NAs, considering the cumulative delay effect from all the potential aggressors may give rise to a conservative $P_{threshold}$ determination. So the percentage bound β is added to control the $P_{threshold}$ metric, permitting experimentation with different crosstalk-induced slowdown along a victim a path and its effect on the crosstalk-pattern count.

Based on these three assumptions, we set the $P_{threshold}$ as the function of process-variation (α), spot delay fault guard band (Δmax), cumulative crosstalk-delay increase, percentage bound (β) and clockcycle (t_{max}), as expressed in the formula given below.

$$P_{threshold}.(1+\ \alpha) + \Delta\max + \beta\,.\sum_{Aggr} \Delta Delay_{crosstalk} = t_{max}$$

A delay-sensitive victim path is considered for crosstalk-induced slowdown if the nominal delay of the path $P_{nominal}$ is above the $P_{threshold}$. That is, whenever the maximum delay of a path under process variation, crosstalk-induced slowdown plus spot delay defect size guard band is less than the clock cycle time t_{max}, the victim path is not considered for cross-induced delay pattern generation. For simplicity, we set Δmax as several gate delays in our experiments.

8. EXPERIMENTAL RESULTS

The proposed path delay test generator maximizing crosstalk-induced slowdown was implemented in Visual C++ and run on a 64-bit Windows 7 PC with Intel Core 2 Duo processor (2.66GHz) and 4GB of memory. Experiments are performed on the ISCAS85 and ISCAS89 benchmark circuits. For our experiments, parasitic information, such as coupling capacitance and load capacitance was extracted using SoC Encounter on TSMC 45nm technology. Net-to-net nominal delays reported in the extracted Standard Delay Format (SDF) file are used for STA delay computation in the crosstalk-induced delay test generator.

8.1 Aggressor Pruning

Table 1 and Table 2 show the results of aggressor pruning for the aggressor-first and pattern-first dynamic compaction algorithms on ISCAS85 benchmark circuits. Column 2 gives the KLPG path count and Column 3 reports the total number of neighboring nets coupled to those paths. We observe from the extracted coupling capacitances of the ISCAS circuits that a substantial number of those neighboring nets coupled to a victim net have insignificant coupling capacitance value. So we compute the potential delay increase of each neighboring net using $\Delta Delay_{crosstalk}$ metric, as detailed in Section 4.3.1.

A minimum delay increase threshold is used thereafter to filter the neighboring nets that have almost no effect on the victim path. The coupled nets with potential delay

increase of less than 2.5% of the victim path delay under test are trimmed off. The 2.5% delay increase threshold is set by analyzing victim path delay increase vs. threshold to determine an appropriate delay vs. cost trade-off. The resultant number of potential aggressors is shown in column 4. For a majority of the ISCAS85 circuits, the 2.5% minimum delay increase threshold reduces the number of aggressors by 75-80%. For larger circuits such as c3540, c5315 and c7552, this delay increase criterion limits the potential aggressors to about 10-15% of the total aggressors extracted from circuit layout. Existing NAs from the victim path forbid some of the aggressors to set an opposite transition on the coupled victim. Column 5 lists aggressors after pruning for victim path NAs. Column 6 reports the number of aggressors that meet timing alignment and transition direction. After pruning, aggressors are inserted into the potential aggressor list for the victim path in decreasing order of *coupling effectiveness*. We consider the values of C1, C2 and C3 as 0.5, 0.25 and 0.25 respectively in the *coupling effectiveness* computation of the aggressor nets. That way we gave more priority to potential delay increase of an aggressor rather than to its timing alignment. So the logical pruning step reduces the potential aggressor candidates by almost half for the ISCAS85 circuits. Of the remaining aggressors, approximately 30% have transition windows that bracket the victim transition and so are considered for timing-aligned crosstalk pattern generation. Table 3 and Table 4 repeat the experiments with a 1% delay increase threshold, with a corresponding increase in number of aggressors.

Table 1 Aggressor pruning in aggr-1st compaction with 2.5% delay th (ISCAS85)

Circuit	# Paths	# Initial Aggr (for all victim paths)	# Aggr meeting Min Delay Increase Th	# Aggr after Logical Pruning	# Aggr with Potential Alignment
c432	312	25036	5127	2282	1119
c499	460	25331	7190	4316	1585
c880	742	38571	11130	6551	1581
c1355	878	81948	14207	5817	2393
c1908	1030	89681	16113	7562	2805
c2670	1464	139298	21825	13665	4105
c3540	1900	285958	26328	11152	5456
c5315	3971	363807	63059	37670	13447
c7552	4633	580903	66361	32145	12205

Table 2 Aggressor pruning in pat-1st compaction with 2.5% delay th (ISCAS85)

Circuit	# Paths	# Initial Aggr (for all victim paths)	# Aggr meeting Min Delay Increase Th	# Aggr after Logical Pruning	# Aggr with Potential Alignment
c432	312	25036	5127	1832	871
c499	460	25331	7190	3633	1315
c880	742	38571	11130	4064	852
c1355	878	81948	14207	5370	2226
c1908	1030	89681	16113	5908	2372
c2670	1464	139298	21825	9253	2580
c3540	1900	285958	26328	9490	5003
c5315	3971	363807	63059	26714	9095
c7552	4633	580903	66361	23342	9089

Table 3 Aggressor pruning in aggr-1st compaction with 1% delay th (ISCAS85)

Circuit	# Paths	# Initial Aggr (for all victim paths)	# Aggr meeting Min Delay Increase Th	# Aggr after Logical Pruning	# Aggr with Potential Alignment
c432	312	25036	12738	4823	2188
c499	460	25331	13897	8089	2653
c880	742	38571	21355	12993	3406
c1355	878	81948	28393	11943	4869
c1908	1030	89681	38024	16782	6571
c2670	1464	139298	46467	29514	9013
c3540	1900	285958	85456	35063	15889
c5315	3971	363807	129157	79394	26723
c7552	4633	580903	164797	89406	33838

Table 4 Aggressor pruning in pat-1st compaction with 1% delay th (ISCAS85)

Circuit	# Paths	# Initial Aggr (for all victim paths)	# Aggr meeting Min Delay Increase Th	# Aggr after Logical Pruning	# Aggr with Potential Alignment
c432	312	25036	12738	3899	1740
c499	460	25331	13897	6982	2180
c880	742	38571	21355	7754	1857
c1355	878	81948	28393	11161	4506
c1908	1030	89681	38024	13457	5696
c2670	1464	139298	46467	20193	6045
c3540	1900	285958	85456	30770	14669
c5315	3971	363807	129157	58175	18747
c7552	4633	580903	164797	63870	24719

8.2 Timing-Oriented ATPG

Table 5 and Table 6 show the crosstalk test generation results for aggressor-first and pattern-first compaction with delay increase threshold of 2.5% and 1% respectively. Further it compares the increase in pattern count with these two compaction approaches. Column 2 lists the compacted test patterns without crosstalk. Columns 3 and 7 compare the number of potential aggressors between aggressor-first and pattern-first compaction. These are the aggressors that meet timing alignment and transition requirements during aggressor pruning steps. Columns 4 and 8 list the number of sensitized aggressors using the two compaction techniques. Similarly columns 5 and 9 show the compacted test patterns with aggressor-first and pattern-first compaction respectively. As we can see from Table 5 and Table 6 aggressor-first compaction can sensitize 60-75% of the potential aggressors from column 3. Sensitizing crosstalk prior to compaction increases pattern count by 150-200% for most of the ISCAS85 benchmark circuits, as shown in column 5. Although there is no increase in pattern count with pattern-first compaction, the NAs of the multiple victim paths in the compacted path set preclude an opposite transition for many aggressors. The NAs also reduce the search space for aggressor sensitization in pattern-first compaction. This leads to an abrupt drop down in the number of sensitized aggressors as shown in column 8 when compared number of aggressors sensitized by aggressor-first compaction in column 4. Columns 6 and 10 further illustrate this by showing the difference in per-path aggressors sensitized by the two compaction techniques. The decrease in sensitized aggressors per victim path in

pattern-first compaction will result in lower crosstalk-induced delay increase. There is clearly a trade-off between pattern count and test quality.

Table 5 Crosstalk pattern generation for ISCAS85 circuits with 2.5% delay th

Circuit	# Comp--acted KLPG Pattern	Aggressor-first Compaction				Pattern-first Compaction			
		# Potential Aggr	# Aggr Justified	# Xtalk Patterns	# Aggr Per Path (Avg)	# Potential Aggr	# Aggr Justified	# Xtalk Patterns	# Aggr Per Path (Avg)
c432	110	1119	781	170	2.5	871	205	110	0.65
c499	265	1585	1194	445	2.59	1315	167	265	0.36
c880	96	1581	1126	210	1.52	852	81	96	0.11
c1355	626	2393	1304	826	1.49	2226	16	626	0.02
c1908	469	2805	1142	661	1.11	2372	4	469	0.004
c2670	280	4105	2901	448	1.98	2580	14	280	0.009
c3540	1107	5456	2230	1579	1.17	5003	78	1107	0.041
c5315	887	13447	9237	1492	2.33	9095	175	887	0.044
c7552	754	12205	5682	1634	1.23	9089	11	754	0.002

Table 6 Crosstalk pattern generation for ISCAS85 circuits with 1% delay th

Circuit	# Comp--acted KLPG Pattern	Aggressor-first Compaction				Pattern-first Compaction			
		# Potential Aggr	# Aggr Justified	# Xtalk Patterns	# Aggr Per Path (Avg)	# Potential Aggr	# Aggr Justified	# Xtalk Patterns	# Aggr Per Path (Avg)
c432	110	2188	1450	231	4.65	1740	448	110	1.44
c499	265	2653	2156	693	4.68	2180	276	265	0.6
c880	96	3406	2304	344	3.11	1857	246	96	0.33
c1355	626	4869	2744	1084	3.13	4506	163	626	0.19
c1908	469	6571	2272	817	2.21	5696	66	469	0.064
c2670	280	9013	6395	783	4.37	6045	141	280	0.096
c3540	1107	15889	5869	2355	3.08	14669	239	1107	0.13
c5315	887	26723	16613	2260	4.18	18747	969	887	0.24
c7552	754	33838	17605	2351	3.79	24719	3824	754	0.83

Table 7 and Table 8 show the results for the aggressor-first and pattern-first dynamic compaction algorithms on ISCAS89 circuits.

Table 7 Crosstalk pattern generation using aggr-1st compaction for ISCAS89 circuits with 1% delay th

Circuit	# Paths	#Aggr meeting Min Delay Incr.Th	# Aggr after Logical Pruning	# Aggr with Potential Alignment	# Aggr Justified	# Patterns	# Xtalk Patterns	CPU Time (s)
s1423	412	22250	8654	3262	910	141	271	141.70
s1488	197	9836	5253	1233	384	70	90	3.67
s1494	199	9240	4853	1131	278	66	84	3.47
s5378	1801	58350	36174	8820	3696	235	531	243.53
s9234	2386	87607	41255	14764	6685	400	1185	1568.7
s13207	3470	103209	44361	13444	5367	870	1056	966.86
s15850	2781	87886	42480	12929	4484	297	438	728.98
s35932	10242	149840	79414	21609	9145	32	161	1748.25
s38417	10640	189315	111959	38726	19840	417	831	4460.20
s38584	10812	170334	98386	25468	12076	285	958	7063.12

Table 8 Crosstalk pattern generation using pat-1st compaction for ISCAS89 circuits with 1% delay th

Circuit	# Paths	# Aggr meeting Min Delay Incr.Th	# Aggr after Logical Pruning	# Aggr with Potential Alignment	# Aggr Justified	# Compact-ed Delay Patterns	# Compact-ed Xtalk Patterns	CPU Time (s)
s1423	412	22250	7142	2679	203	141	141	100.11
s1488	197	9836	3568	817	28	70	70	2.55
s1494	199	9240	3178	720	56	66	66	2.95
s5378	1801	58350	24411	6554	411	235	235	65.63
s9234	2386	87607	27756	9971	154	400	400	1054.32
s13207	3470	103209	29329	9387	70	870	870	633.4
s15850	2781	87886	26924	9237	18	297	297	482.51
s35932	10242	149840	43148	12708	24	32	32	811.80
s38417	10640	189315	105638	38501	540	417	417	5326.74
s38584	10812	170334	54625	14960	46	285	285	1540.92

8.3 ATPG Run-Time Overhead

Table 9 lists the CPU time for each component of the proposed crosstalk-driven pattern generation algorithm using a 1% delay threshold. Column 2 shows the potential aggressors that meet the minimum potential delay increase and timing alignment requirements. The number of aggressors for which a path from the PIs cannot be found is listed in column 3. Column 4 lists how many complete paths failed justification. The CPU time required to generate the victim paths is shown in column 5. Column 6 shows the CPU time for pruning the initial aggressor candidates. Column 7 lists the CPU time to generate aligned patterns for the potential aggressors and dynamically compact those to maximize the effects of crosstalk-induced delay on a target path. As can be seen, for all the benchmark circuits, the aggressor pruning step takes little time. Most of the time is either spent in victim path generation or aggressor sensitization. c499, c3540, c5315 and c7552 benchmark circuits spend most of the CPU time targeting aggressors, while c432, c1908, c2670 circuits take less time for aggressor path sensitization. The amount of time in aggressor sensitization is dominated by the number of aggressors that fail sensitization or justification in columns 3 and 4. Since justification is the most expensive step in victim path and aggressor path generation, the benchmarks with more aggressor paths failing final justification spend more time in generating aggressor transitions. So speeding up the algorithm is mostly dependent on using a faster justification procedure. Table 10 shows the CPU time for pattern-first compaction. The increased NAs filter out more aggressors, so there are fewer justification failures and so much lower CPU time.

Table 9 CPU time breakdown for aggr-1st compaction (ISCAS85)

Circuit	#Potential Aggr	#Aggr Not Sensitized	#Aggr Not Justified	CPU Time (s) Path Gen.	CPU Time (s) Aggr Pruning	CPU Time (s) Xtalk Pat Gen
c432	2188	738	99	44.91	0.38	9.20
c499	2653	497	703	2.04	0.66	139.81
c880	3406	1102	2	0.72	1.11	10.99
c1355	4869	2125	589	45.87	1.96	346.13
c1908	6571	4299	564	346.69	1.90	163.19
c2670	9013	2618	112	85.41	3.37	71.17
c3540	15889	10020	5094	287.11	5.88	4312.90
c5315	26723	10110	2702	54.73	11.26	511.16
c7552	33838	16233	13910	217.36	14.83	1219.89

Table 10 CPU time breakdown for pat-1st compaction (ISCAS85)

Circuit	#Potential Aggr	#Aggr Not Sensitized	#Aggr Not Justified	CPU Time (s) Path Gen.	CPU Time (s) Aggr Pruning	CPU Time (s) Xtalk Pat Gen
c432	1740	1292	0	45.94	0.73	7.04
c499	2180	1904	99	1.78	1.84	54.40
c880	1857	1611	12	0.98	0.27	0.20
c1355	4506	4343	4	47.65	4.48	191.55
c1908	5696	5630	13	348.69	3.63	16.03
c2670	6045	5904	1367	72.56	4.22	25.74
c3540	14669	14430	168	286.18	10.66	393.15
c5315	18747	17778	144	53.53	13.71	404.95
c7552	24719	20895	221	216.37	16.13	49.71

8.4 Timing-Oriented ATPG with Low Cost Fault Coverage Metric

Experiments on aggressor-first compaction were conducted to demonstrate the benefits of timing-oriented ATPG with the low cost delay fault coverage metric. We performed experiments onISCAS85 benchmark circuits. The clock period is set to be 8% longer than the nominal delay of the longest testable path. It is assumed that there is only one spot delay defect in any target victim path and the circuit is subject to process variation. For the low cost aggressor-first compaction experiments, the crosstalk delay increase threshold is considered as 1% of the victim path delay.

In the first experiment, process variation is assumed to be ±20% of the nominal path delay (α) and the local random spot defect guard band (Δmax) is 3 gate delays. We assume that local delay defects exceeding 3 gates are essentially transition faults. In our crosstalk ATPG environment, once a victim path is generated, we prune away the aggressor candidates that do not meet the delay increase and timing alignment requirements. We compute the cumulative delay increase from the remaining potential aggressors coupled to that victim path for use in low-cost metric. However, a percentage of the potential aggressors cannot be sensitized in the presence of victim path NAs, so considering the cumulative delay effect from all the potential aggressors may give rise to a conservative $P_{threshold}$ determination. We can observe from columns 3 and 4 in Table 5 and Table 6 that aggressor-first dynamic compaction can sensitize approximately 75% of the potential aggressors for most benchmark circuits. However, for circuits like c3540, c5315 and c7552, the percentage of aggressors justified is about 50%. It is quite

likely that some victim paths have many more aggressors sensitized than others. Setting the percentage bound β to 0.5 can cause crosstalk pattern generation to erroneously skip some victim paths. We use a β of 0.75 in our experiments and accordingly set the $P_{threshold}$ value for different victim paths in ISCAS85 circuits.

Table 11 shows the results of aggressor-first compaction using the low cost fault coverage metric. Column 2 lists the number of delay-sensitive victim paths that are considered for cross-induced delay pattern generation. Column 3 shows the total number of aggressors that meet timing alignment and transition direction on those victim paths considered for crosstalk pattern generation in column 2. The number of sensitized aggressors is listed in column 4. Column 5 shows the compacted patterns count with the low-cost fault coverage metric. For c5315 and c7552, the number of victim paths considered for timing-driven ATPG is small, which indicates that many fault sites are dropped because the longest paths through them are short. In c1355, many paths are considered for crosstalk ATPG. This is because this circuit is optimized to have many paths close to the maximum delay.

In the second experiment, process variation is set to $\pm$30%. The local delay defect guard band is kept at 3 gate delays and crosstalk percentage bound β is 0.75. Table 12 shows the results. Since $P_{threshold}$ is decreased, more victim paths with shorter nominal length will be considered for crosstalk-induced delay pattern generation. The number of test vectors is sensitive to the parameters interacting with the circuit path delay distribution.

Table 11 Aggr-1st compaction with low cost coverage metric for ISCAS85 circuits (with 20% process variation)

Circuit	# Paths Considered	# Potential Aggr	# Aggr Justified	#Compacted Crosstalk Patterns
c432	202	1637	1028	193
c499	396	2491	1995	682
c880	222	1145	682	204
c1355	616	4018	2155	1037
c1908	450	3444	853	579
c2670	546	3888	2379	544
c3540	960	8296	2650	1721
c5315	246	1671	403	899
c7552	347	2137	576	864

Table 12 Aggr-1st compaction with low cost coverage metric for ISCAS85 circuits (with 30% process variation)

Circuit	# Paths Considered	# Potential Aggr	# Aggr Justified	#Compacted Crosstalk Patterns
c432	219	1724	1081	201
c499	396	2491	1995	682
c880	266	1421	860	231
c1355	701	4323	2341	1076
c1908	501	3855	931	586
c2670	603	4162	2555	555
c3540	1059	9112	2924	1794
c5315	346	2319	566	910
c7552	487	3076	893	927

Table 13 compares the crosstalk test size using low cost coverage metric to the regular crosstalk test with α, β and Δmax as 20%, 0.75 and 3 gate delays respectively. Columns 2 and 5 compare the number of victim paths considered for crosstalk pattern generation for the two tests. Columns 3 and 6 compare the CPU time for the two tests. Column 7 shows the decrease in compacted pattern count with low cost coverage metric in comparison to column 4. Column 8 reports the speedup factor. Overall, aggressor-first compaction is much faster with low cost coverage metric. With the implementation of the low-cost coverage metric, the aggressor-first compaction has a much smaller test size with reasonable CPU time overhead for crosstalk pattern generation.

Table 13 Crosstalk pattern count comparison

Circuit	Without Low Cost Coverage Metric			With Low Cost Coverage Metric			Speed Up Factor
	# Paths Considered	ATPG Time (s)	# Compacted Patterns	# Paths Considered	ATPG Time (s)	# Compacted Patterns	
c432	312	55.54	231	202	53.59	193	1.03
c499	460	142.52	693	396	141.78	682	1.00
c880	742	12.83	344	222	8.97	204	1.43
c1355	878	413.97	1084	616	394.23	1037	1.05
c1908	1030	513.79	817	450	451.67	579	1.13
c2670	1464	159.96	783	546	137.19	544	1.16
c3540	1900	4605.90	2355	960	3121.68	1721	1.47
c5315	3971	577.17	2260	246	98.85	899	5.83
c7552	4633	1452.11	2351	347	386.94	864	3.75

8.5 Crosstalk ATPG for Non-Robust and Long Transition Test

The prior results were generated using robust sensitization for the victim path and the path to each aggressor site. Table 14 shows the results for crosstalk pattern generation for all testable paths on ISCAS89 circuits. In this experiment, KLPG test generation is used to generate the longest path through each line under robustness constraints, topped off with non-robust path tests for the dropped target paths, topped off with long transition fault tests. The results further show the increase in testable paths, pattern count, number of sensitized aggressors and CPU time with top-off tests. Columns 2 and 6 in Table 14 show that top-off tests increase the testable paths by 100-300% for most of the ISCAS89circuits except s5378 and s35932. That results in increase in number of sensitized aggressors, as listed in columns 3 and 7 respectively. However, this increase in testable paths and sensitized aggressors inflate the crosstalk pattern count by about 10-60%, as shown in columns 4 and 8 respectively. Columns 5 and 9 compare the CPU time between robust and top-off tests. For a majority of the benchmark circuits, top-off tests increase the crosstalk pattern generation time by about 300-400%. However, for s5378 and s35932, top-off tests double the CPU time, as non-robust and long-transition tests cannot increase the testable paths substantially on top of robust tests.

Table 14 Crosstalk pattern generation for all testable paths on ISCAS89 circuits

Circuit	Robust				Robust + Non-robust + Long-transition			
	# Testable Paths	# Aggr Sensitized	# Xtalk Patterns	CPU Time (s)	# Testable Paths	# Aggr Sensitized	# Xtalk Patterns	CPU Time (s)
s1423	412	910	271	141.70	790	1281	363	575.01
s1488	197	384	90	3.67	649	1138	161	15.45
s1494	199	278	84	3.47	650	946	151	15.03
s5378	1801	3696	531	243.53	1995	4038	577	431.10
s9234	2386	6685	1185	1568.76	3583	9277	1415	10063.0
s13207	3470	5367	1056	966.86	6132	8055	1670	4726.22
s15850	2781	4484	438	728.98	5045	5766	491	2810.39
s35932	10242	9145	161	1748.25	11730	9525	171	3608.23
s38584	10812	12076	958	7063.12	17021	17392	1255	7779.97

9. COMPARISON AND CORRELATION

In order to verify that the crosstalk-aware patterns generated by our timing-driven test generator maximize crosstalk-induced delay on delay-sensitive paths, we will compare their delay distribution against the path delays for zero-filled and random-filled KLPG patterns ignoring crosstalk. In addition, we will show the correlation between the estimated crosstalk-induced delay increase using our proposed $\Delta Delay_{crosstalk}$ metric and the delay increase observed from circuit simulation of the crosstalk patterns. The estimated delay increase may deviate from the simulated one as the $\Delta Delay_{crosstalk}$ metric does not take into account the fortuitous helper transitions or aligned aggressors coupling to the same victim net.

9.1 Crosstalk Delay Increase

Table 15 compares the circuit simulation delay of 20 randomly selected testable paths in c5315 using zero-filled, random-filled and crosstalk delay-induced KLPG pattern. These 20 paths have experienced increase in delay only because of crosstalk coupling. Column 2 shows the path delay for a zero-filled conventional KLPG pattern ignoring crosstalk. Column 3 reports the delay increase of the same KLPG pattern whose unspecified bits are random-filled. We repeat each and every random-filled KLPG simulation 10 times with different random values for the unspecified bits. The delay increase from our proposed alignment-driven crosstalk patterns are shown in column 4. Column 5 lists the percentage increase in path delay using our crosstalk patterns when

compared to the zero-filled path delay. We can see from Table 15 that for all the victim paths selected (except path no. 18), the crosstalk patterns have longer delay than the zero-filled and random-filled KLPG patterns. Columns 6, 7 and 8 in Table 15 further confirm this by comparing the number of aggressors sensitized by the three pattern set. Random-filling of the unspecified bits in path no. 18 have created additional fortuitous aligned aggressors coupling to the same path, which results in more delay increase than crosstalk patterns. For few paths in Table 15, random-filling reduce path delays than zero-filled path delays ignoring crosstalk. This is due to helper transitions being accidentally generated by the random-filling of unspecified bits.

Figure 10 further shows the delay increase using our crosstalk delay-induced patterns for 45 testable paths spread over the entire path delay distribution of c5135. In order to show the impact of crosstalk-induced slowdown from our timing-driven ATPG, we did the circuit simulations of those 45randomly chosen paths, that have a potential delay increase of more than 15ps and the number of sensitized aggressors coupled to those victim paths are at least 4. Those 45 paths are simulated in turn using zero-filled, random-filled KLPG patterns ignoring crosstalk and crosstalk patterns from our timing-oriented ATPG.

Table 15 Crosstalk delay increase for c5315

Path No.	Path Delay (0-filled KLPG) (ps)	Increase in Delay (ps)		% Incr in Delay by Xtalk Pattern	# Aligned Aggressors Sensitized		
		Random-filled KLPG	Xtalk Pattern		0-filled KLPG	Random-filled KLPG	Xtalk Pattern
1	364.84	-4.15	4.41	1.21	0	0	6
2	425.38	4.54	9.53	2.24	0	4	6
3	447.81	2.57	8.38	1.87	0	1	5
4	453.69	5.37	12.51	2.75	0	4	7
5	462.69	-3.91	8.39	1.81	0	-2	6
6	474.49	-3.81	12.12	2.55	0	0	5
7	475.51	0	4.03	0.84	0	1	6
8	513.15	1.05	8.01	1.56	1	1	5
9	530.19	0	10.63	2.00	1	1	8
10	563.05	3.09	4.98	0.88	1	1	3
11	604.64	5.31	10.34	1.71	0	1	3
12	620.97	0	8.83	1.42	0	0	3
13	627.67	7.03	10.85	1.72	0	2	3
14	641.51	3.57	10.83	1.68	-2	1	2
15	700.83	2.77	4.53	0.64	0	2	3
16	725.4	0.31	3.84	0.52	0	0	4
17	765.74	-2.51	3.99	0.52	0	0	2
18	816.51	14.78	9.01	1.10	-3	4	5
19	880.93	3.13	5.91	0.67	0	2	2
20	883.15	1.05	8.01	0.90	1	1	5

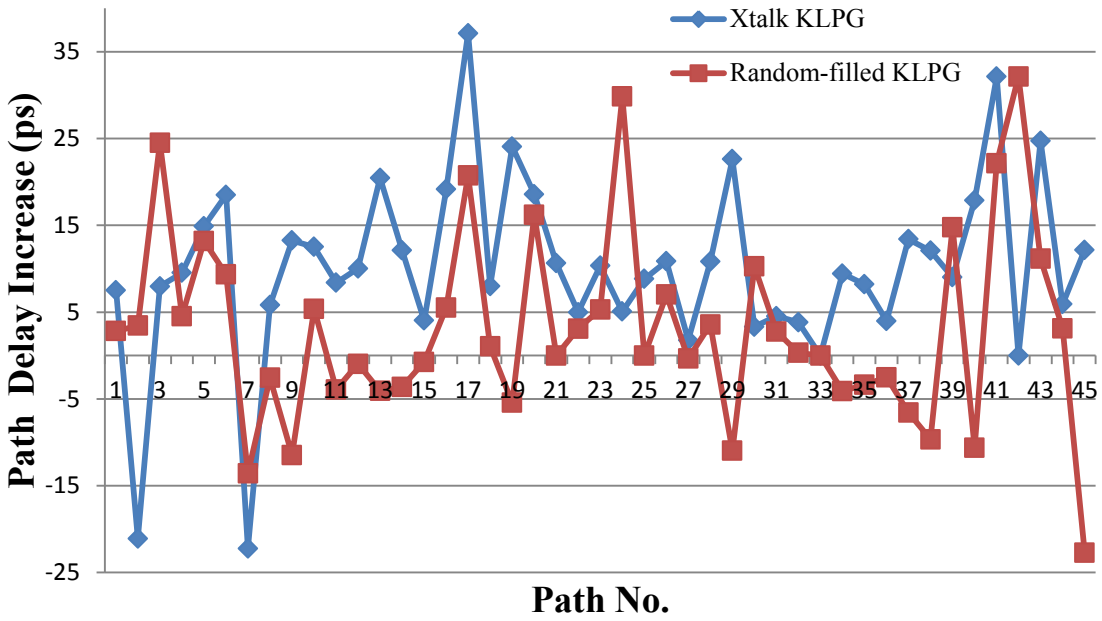

Figure 10 Increase in path delay for c5315

We can see from Figure 10 that for the majority of victim paths selected, the alignment-driven crosstalk patterns have longer delay than the zero-filled and random-filled KLPG patterns. However, there are quite a few paths in Figure 10, where the crosstalk-aware patterns are not slower than the zero-filled or random-filled patterns. The reasons for this are discussed in the following sections.

9.1.1 Delay at the Side-Input Transition during Robust Test

According to the definition of a robust test, the KLPG ATPG engine satisfies the following conditions to guarantee the detection of a delay fault regardless of the delays of all other gates:

1. For propagating a to-controlling transition at an on-path input, the side fan-ins must be set to their static non-controlling value.
2. For propagating a to-non-controlling transition at an on-path input, the side fan-ins must also have to-non-controlling transitions.

As the second condition of robust test allows to-non-controlling transitions at the side fan-ins, a late transition at side fan-ins can delay the propagation of the on-path input transition and thus potentially can change the on-path timing down the path. A delay fault is still detected, but not on the victim path. Thus detailed timing information at the side inputs is required to ensure that the generated test for an on-path transition is not affected by the side fan-ins. Due to the impracticality of using such information in an ATPG, the KLPG ATPG tool does not take into account the timing at the side fan-ins. In our proposed timing-driven crosstalk ATPG, we attempt to sensitize the potential aggressors once a victim path and the timing on the on-path nets is completely known. We leverage the timing of the victim nets to align the coupled aggressors. However, it is observed during circuit simulations of the crosstalk-driven patterns that sometimes the side-input non-controlling transitions slow down the on-path transition. This in turn changes the victim net timing that our timing-driven ATPG has utilized to align the coupled aggressors down the path. This is entirely circuit-specific. For some of the testable paths such as path 27, 33 and 42 for c5315 in Figure 10, the side-input transitions change the timing of the victim paths and so the crosstalk patterns cannot

align the aggressor transitions to the victim net transitions. As a result, they do not experience any delay from coupling noise.

In addition, for some of the testable paths like path 2, 3, 7, 24, 42 in Figure 10, the zero-filled and random-filled patterns have higher delay than the crosstalk-aware pattern. Zero-filling or random-filling of the unspecified bits in those patterns delays the side-input transitions and thus slows down the propagation of the overall victim path.

9.1.2 Mismatch in Cell Characterization between SDF and SPICE

In our timing-driven crosstalk ATPG, we use the net-to-net nominal delays reported in the extracted SDF file for all timing analysis. The STA engine uses those delay values to compute the earliest and latest possible rising/falling transition timing windows during alignment-driven aggressor net sensitization. The net-to-net delay values in SDF are computed using cell delay-lookup table in timing library file, which contains the delay values for various input slew rates and output load capacitances. So the delay values obtained from SDF file are conservative. We observe that the path delays obtained in ISCAS85 circuits using circuit simulation are approximately 25-30% less than the path delays estimated using SDF data. As our crosstalk-driven ATPG uses SDF data for aligning an aggressor transition with victim net, it is quite likely that the crosstalk patterns from our timing-driven ATPG cannot generate aligned transitions in circuit simulations, due to this mismatch between circuit simulation delay and SDF reported delay. However, we have observed that for majority of cases the delays of the victim path and the aggressor paths are affected equally in circuit simulations. As a

result, most of our crosstalk-aware patterns are still effective in generating aligned transitions in SPICE simulations. Further in order to reduce the mismatch in cell characterization between SPICE and SDF, we perform the SPICE simulations at an elevated temperature of 55C.

Figure 11, Figure 12 and Figure 13 compare the change in delays by our crosstalk patterns against the path delays induced by zero-filled and random-filled KLPG patterns for c2670, c7552 and c1355 respectively.

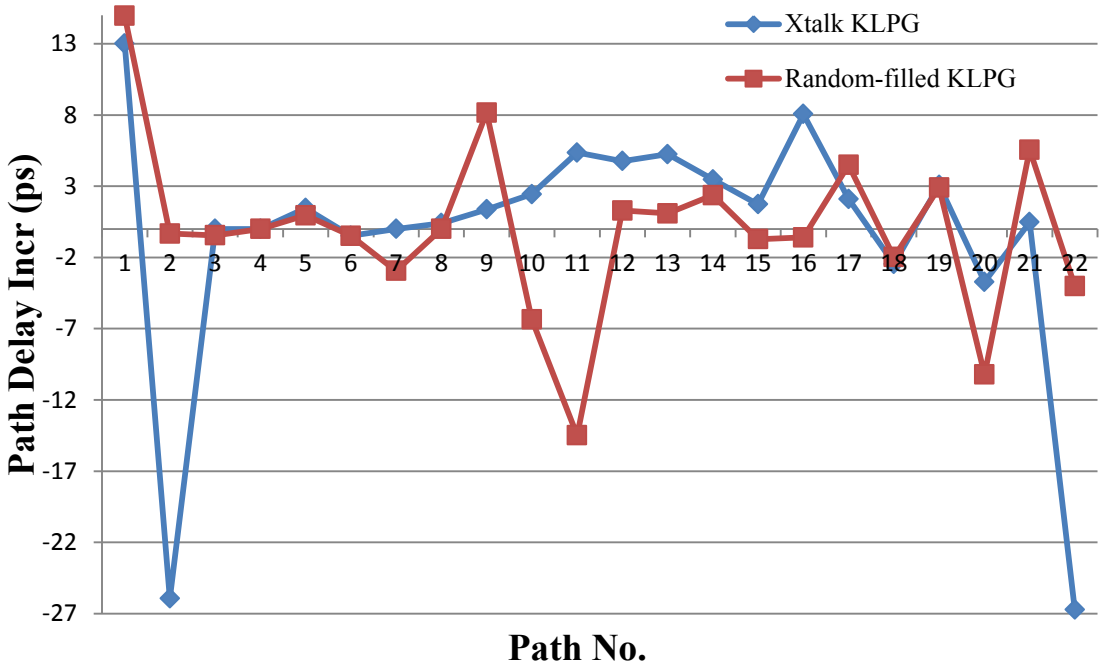

Figure 11 Increase in path delay for c2670

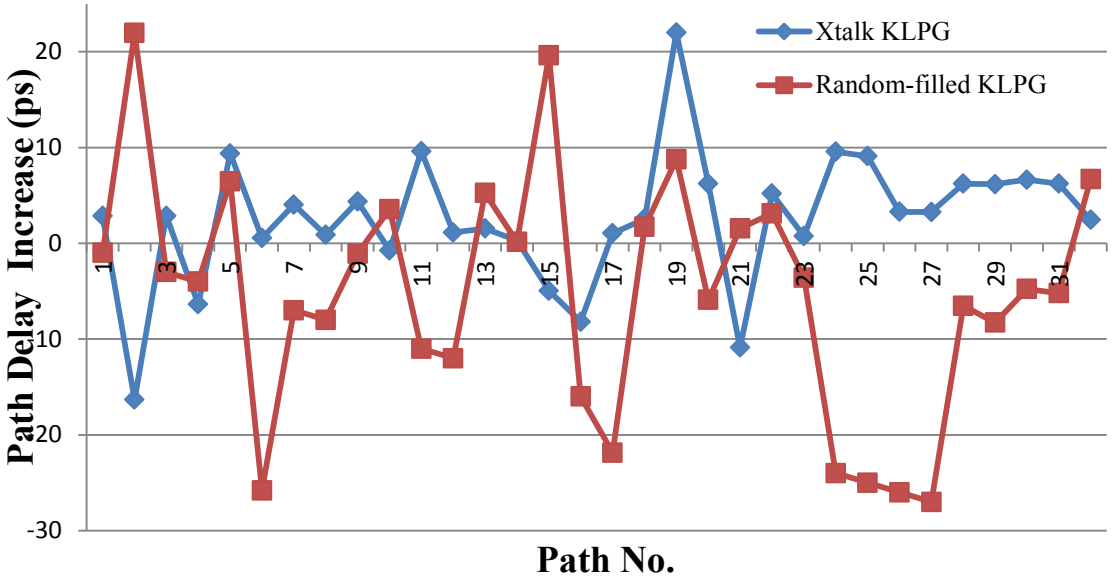

Figure 12 Increase in path delay for c7552

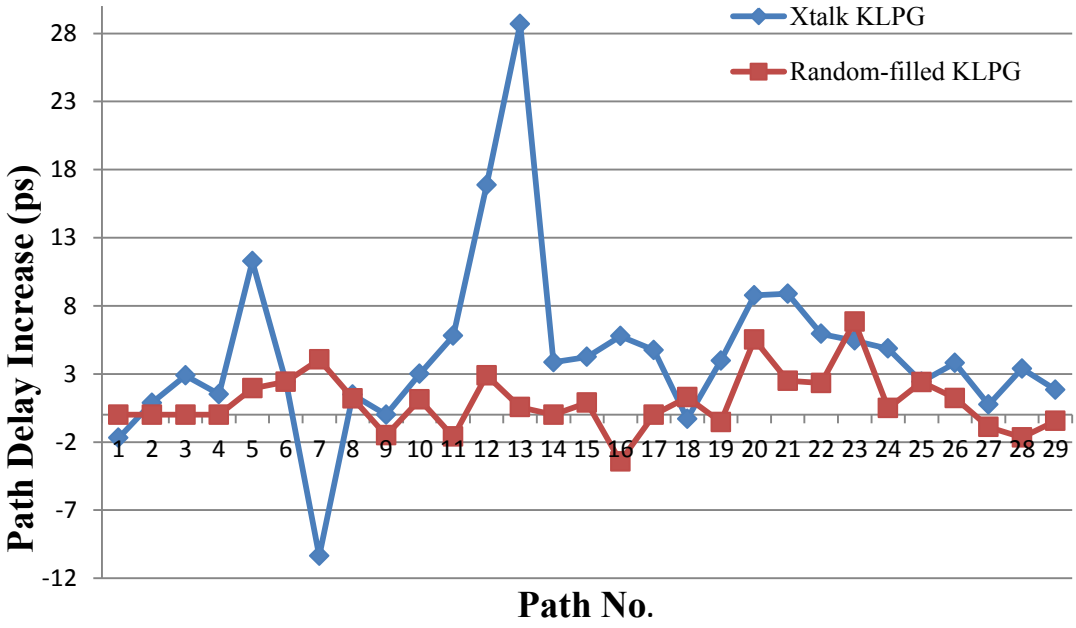

Figure 13 Increase in path delay for c1335

9.2 Estimated vs. Observed Crosstalk Delay Increase

We conducted experiments to determine the correlation between our estimated delay increase metric and the delay increase observed using SPICE simulations of our crosstalk patterns. Figure 14 shows the correlation between estimated and observed delay increase for 45 randomly selected paths in c5315 circuit. We can see from Figure 14 that estimated delay increase sets an upper bound for observed delay increase. For the majority of the paths, the alignment-driven crosstalk patterns could not induce substantial delay increase, as expected from the estimated delay increase metric.

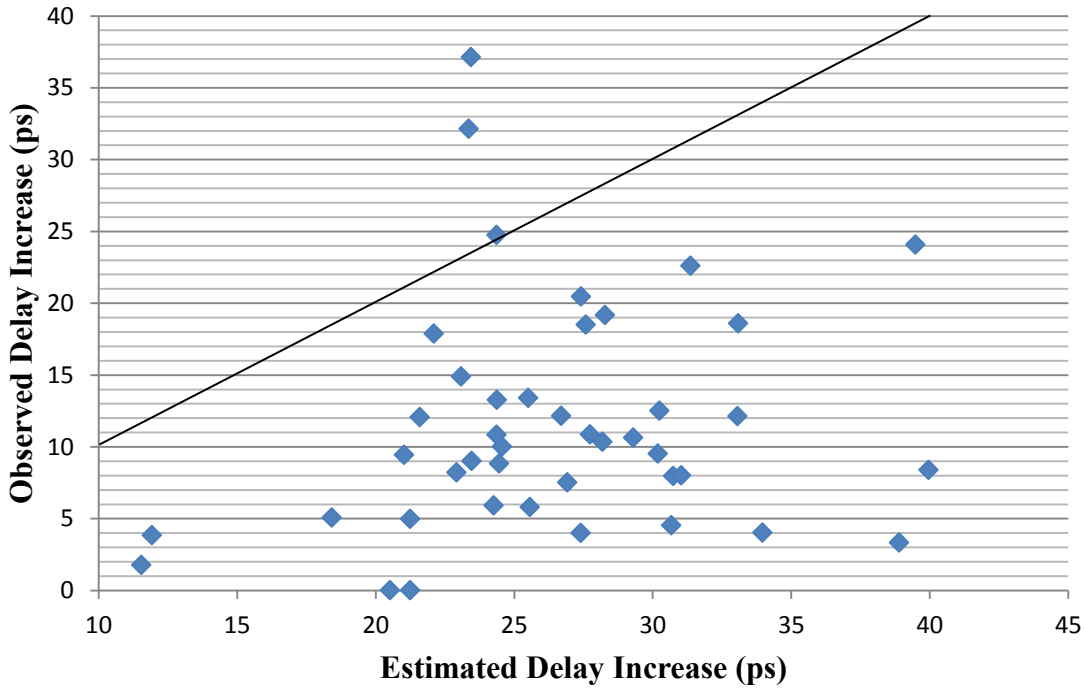

Figure 14 Correlation between estimated and observed delay increase for c5315

We can see from Figure 14 that for a few testable in c5315 circuit, the crosstalk patterns induce more delay than the estimated $\Delta Delay_{crosstalk}$ metric. As the $\Delta Delay_{crosstalk}$ metric is the upper bound for the observed crosstalk delay increase, one reason behind this additional delay is that the $\Delta Delay_{crosstalk}$ metric does not take into account any fortuitous aligned aggressor couplings to the same victim paths. Further the additional necessary assignments generated from the multiple aggressor sensitizations may delay the side-input non-controlling transitions of the target path. This is turn can increase the victim path delay by more than the estimated value. Once the late side-inputs change victim path timing, these late inputs in turn destroy the alignment that timing-driven ATPG has assumed for sensitizing the aggressors coupled to that path. So a delay increase of more than the estimated $\Delta Delay_{crosstalk}$ metric may be due to late side-input transitions being accidently generated from the aggressor NAs.

For a majority of the paths, the timing-driven crosstalk patterns could not induce substantial delay increase, compared to the prediction by the estimated delay increase metric. One reason behind this is the mismatch between circuit simulation delay and SDF reported delay as detailed in Section 9.1.2. As a result, in circuit simulations of the crosstalk patterns, the aggressors lose alignment to victim transitions and thus incur no increase in victim net delay. The other reason may be the assumptions involved in crosstalk delay-induced modeling.

9.2.1 Assumptions involved in $\Delta Delay_{crosstalk}$ Metric

In our crosstalk-induced delay modeling detailed in Section 3, we assume the aggressor and victim nets have completely overlapping transitions and so we do not take into account the impact of skew in crosstalk delay estimation. However, we observe in circuit simulation of the crosstalk patterns that if the aggressor transition is skewed by as little as 50ps, the aggressor net will appear quiescent as the transition propagates through the targeted path. In addition, there are three additional objectives in creating a crosstalk effect of large severity: a weak driver on the victim line, a fast signal transition on the affecting line and a propagation path that maintains or amplifies the noise effect until it reaches an output. But our crosstalk-delay model does not consider the slew rates of the aggressor and victim nets in delay increase estimation. So the estimated delay increase is quite conservative in nature. All these above reasons lead to a poor correlation between the estimated and observed delay increase in circuit c5315.

The model approximates the delay as linear in the change in C_{eff}. Further we do not consider the impact of aggressors on each other to compute the potential delay increase on the victim net. Moreover, the afore-mentioned delay model considers the nominal stage delay of the victim net $Delay_{driver-stage}$ in the crosstalk-delay increase computation. However, a change in input-signal slope caused by crosstalk can impact the nominal stage delay at its receiving gates. The impact of noise on a signal line may affect its receiver gates differently because of varying switching threshold across those.

Figure 15 shows the correlation between the estimated and observed delay increase for circuit c7552.

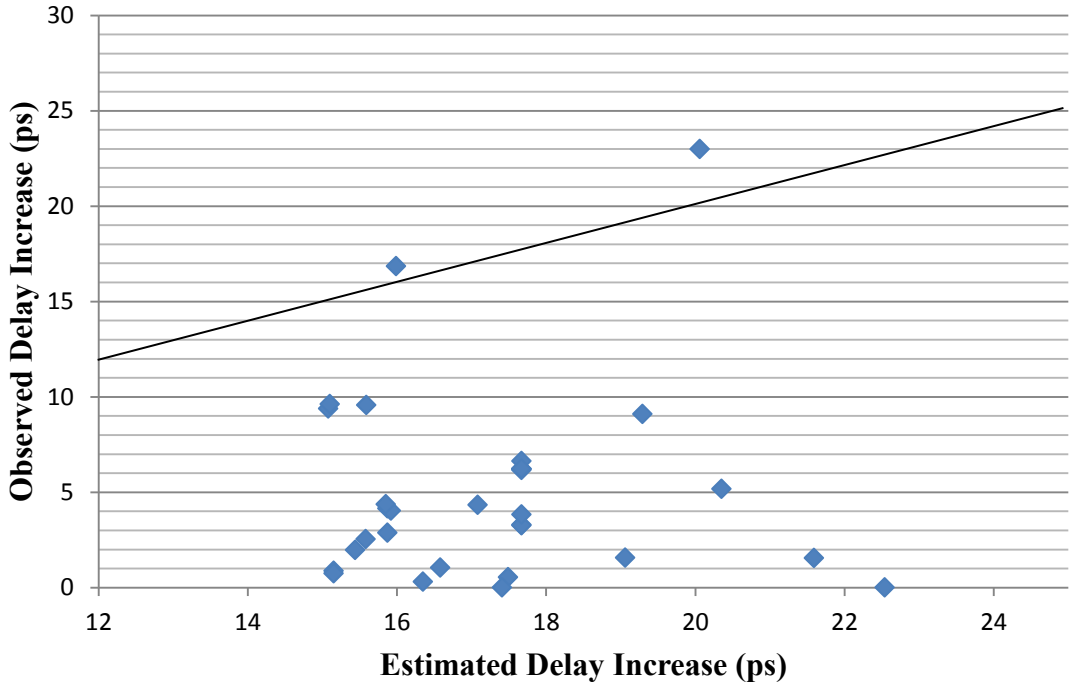

Figure 15 Correlation between estimated and observed delay increase for c7552

9.3 Crosstalk Test under Non-Robust and Long Transition Constraints

Although shows the increase in number of sensitized aggressors with non-robust and long transition tests, the alignment of aggressor transitions with victim path is even more uncertain in comparison to robust test. As the non-robust test allows controlling values at side fan-ins in the first time frame, so a late transition at side fan-ins can block

the propagation of the on-path transition, which is targeted for crosstalk delay increase. As a result, the crosstalk delay effect cannot reach an observable output through the victim path. In addition, more than one transition can attempt to propagate through a target path and this in turn can change the target path timing that our test generator has utilized to align coupled transitions down that path. Further the transitions at side fan-ins may create fortuitous helper transitions along a target path. In addition, the effects of hazards and glitches can interfere with the observation of the output value.

10. CONCLUSION AND FUTURE WORK

In this work, we have proposed a novel timing-oriented test generation algorithm to target multiple aligned aggressors coupled to a target victim path to maximize the crosstalk slowdown effects. The algorithm utilizes timing windows, potential delay increase, and logic constraints to prune a substantial number of ineffective aggressor couplings and thus speed up the pattern generation process. In addition, this algorithm introduces the concept of alignment-driven path sensitization to generate timing-aligned crosstalk patterns. As this test generator sensitizes aggressors in the presence of victim path NAs, the search space is effectively reduced for aggressor path generation. It helps in reducing the crosstalk pattern generation time for aggressors. This algorithm was applied with aggressor-first and pattern-first dynamic compaction.

There are several open issues to be addressed in the future course of this work. The current approach does not consider the fact that due to process variation and side-input transition delays, the timing alignment is uncertain. In general there are many different paths to a coupled line, so it is possible to sensitize a coupled transition at a number of different times. To thoroughly test a circuit, it is necessary to generate test patterns that sweep the coupled transition across a range large enough that all cases of potential alignment are considered. Here in this work we have used the nominal delays for victim path transition. In the future work, we will set a delay bound around the on-path

transition and then attempt to justify up to *M* different paths that cover that range. Min/max path delays will limit the search space.

If two potential aggressors have almost equal *coupling effectiveness*, sensitizing one aggressor first may preclude a greater number of aligned transitions along a victim path due to conflicting logical and timing constraints. So the greedy algorithm may miss the worst possible crosstalk delay increase. In our experience, a greedy algorithm will come close to achieving the worst-case delay increase, particularly when considering the fact that timing alignment is uncertain due to side-input transitions and intra-die process variation. To quantify the impact of a greedy algorithm requires implementation of an exact algorithm.

In the *coupling effectiveness* metric, we have assumed a single value for parameters C1, C2 and C3 and generated crosstalk patterns accordingly. The chosen parameters gave priority to potential delay increase of an aggressor rather than to its timing alignment. In future work, a sensitivity analysis of these parameters must be performed.

In this work, we have demonstrated the change in path delay using our alignment-driven crosstalk patterns against the path delays induced by zero-filled and random-filled KLPG patterns ignoring crosstalk. We further intend to compare the crosstalk delay increase by our approach against prior work [10] that targeted multiple aggressors along a victim path, but did not consider any timing alignment.

REFERENCES

[1] W. Chen, S. Gupta, and M. Breuer, "Analytic models for crosstalk delay and pulse analysis under non-ideal inputs," in *Proc. Int. Test Conf.*, Nov. 1997, pp. 809–818.

[2] R. Anglada and A. Rubio, "Logic fault model for crosstalk interferences in digital circuits," *Int. J. Electron.*, vol. 67, no. 3, pp. 423–425, Feb. 1989.

[3] F. Moll and A. Rubio, "Spurious signals in digital CMOS VLSI circuits : a propagation analysis," *IEEE Trans. Circuits Syst.*, vol. 39, no. 10, pp. 749–752, Oct. 1992.

[4] W. Chen, S. Gupta, and M. Breuer, "Test generation for crosstalk-induced delay in integrated circuits," in *Proc. Int. Test Conf.*, Sep. 1999, pp. 191–200.

[5] S. Kundu, S. T. Zachariahi, Y. Chang, and C. Tirumurti, "On modeling crosstalk faults," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 12, pp. 1909–1915, Dec. 2005.

[6] A. Rubio, N. Itazaki, X. Xu, and K. Kinoshita, "An approach to the analysis and detection of crosstalk faults in digital VLSI circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 3, pp. 387–395, Mar. 1994.

[7] N. Itazaki, Y. Matsumoto, and K. Kinoshita, "An algorithmic test generation method for crosstalk faults in synchronous sequential circuits," in *Proc. Asian Test Symp.*, Nov. 1997, pp. 22–27.

[8] W. Chen, S. Gupta, and M. Breuer, "Test generation in VLSI circuits for crosstalk

noise," in *Proc. Int. Test Conf.*, Oct. 1998, pp. 641–650.

[9] K. T. Lee, C. Nordquist, and J. A. Abraham, "Automatic test pattern generation for crosstalk glitches in digital circuits," in *Proc. VLSI Test Symp.*, Apr. 1998, pp. 34–39.

[10] J. Lee and M. Tehranipoor, "A novel pattern generation framework for inducing maximum crosstalk effects on delay-sensitive paths," in *Proc. Int. Test Conf.*, Oct. 2008, pp. 1–10.

[11] K. P. Ganeshpure and S. Kundu, "On ATPG for multiple aggressor crosstalk faults," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 5, pp. 774–787, May 2010.

[12] S. Irajpour, S. K. Gupta, and M. A. Breuer, "Timing-independent testing of crosstalk in the presence of delay producing defects using surrogate fault models," in *Proc. Int. Test Conf.*, Oct. 2004, pp. 1024–1033.

[13] K. P. Ganeshpure and S. Kundu, "Automatic test pattern generation for maximal circuit noise in multiple aggressor crosstalk faults," in *Proc. Des. Autom. Test Eur.*, Apr. 2007, pp. 1–6.

[14] M. Palla, J. Bargfrede, K. Koch, W. Anheier, and R. Drechsler, "Adaptive branch and bound using SAT to estimate false cosstalk," in *Proc. Int. Symp. Quality Electronic Des.*, Mar. 2008, pp. 508–513.

[15] M. Palla, J. Bargfrede, S. Eggersgl¨uß, and W. Anheier, "Timing arc based logic

analysis for false noise reduction," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 2009, pp. 225–230.

[16] S. Chun, T. Kim, and S. Kang, "ATPG-XP: test generation for maximal crosstalk-induced faults," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 9, pp. 1401–1413, Sep. 2009.

[17] H. Li and X. Li, "Selection of crosstalk-induced faults in enhanced delay test," *J. Electron. Test.*, vol. 21, no. 2, pp. 181–195, Apr. 2005.

[18] H. Li, Y. Zhang, and X. Li, "Delay test pattern generation considering crosstalk-induced effects," in *Proc. Asian Test Symp.*, Nov. 2003, pp. 178–183.

[19] X. Bai, S. Dey, and A. Krstic, "HyAC: a hybrid structural SAT based ATPG for crosstalk," in *Proc. Int. Test Conf.*, Sep. 2003, pp. 112–121.

[20] A. Krstic, J. Liou, Y. Jiang, and K. T. Cheng, "Delay testing considering crosstalk-induced effects," in *Proc. Int. Test Conf.*, Oct. 2001, pp. 558–567.

[21] B. C. Paul and K. Roy, "Testing crosstalk-induced delay faults in static CMOS circuit through dynamic timing analysis," in *Proc. Int. Test Conf.*, Oct. 2002, pp. 384–390.

[22] Y. Ran, A. Kondratyev, K. H. Tseng, Y. Watanabe, and M. Marek-Sadowska, "Eliminating false positives in crosstalk noise analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 9, pp. 1406–1419, Sep. 2005.

[23] Ke Peng, M. Yilmaz, M. Tehranipoor, and K. Chakrabarty, "High-quality pattern

selection for screening small-delay defects considering process variations and crosstalk," in *Proc. Des. Autom. Test Eur.*, Mar. 2010, pp. 1426–1431.

[24] J. Kong, D. Z. Pan, and P. V. Srinivas, "Improved crosstalk modeling for noise constrained interconnect optimization," in *Proc. Asia South Pacific Des. Autom. Conf.*, Jan. 2001, pp. 373–378.

[25] M. Becer, V. Zolotov, R. Panda, A. Grinshpon and I. Algol, "Pessimism reduction in crosstalk noise aware STA," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 2005, pp. 954–961.

[26] M. Kulkarni and T. Chen, "A sensitivity-based approach to analyzing signal delay uncertainty of coupled interconnects," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 9, pp. 1336–1346, Sep. 2005.

[27] B. Choi and D. M. H. Walker, "Timing analysis of combinational circuits including capacitive coupling and statistical process variation," in *Proc. VLSI Test Symp.*, Apr. 2000, pp. 49–54.

[28] T. Rajeshwary and J. A. Abraham, "Critical path selection for delay test considering coupling noise," in *Proc. Eur. Test Symp.*, May 2008, pp. 119–124.

[29] T. Sato, Y. Cao, D. Sylvester, and C. Hu, "Characterization of interconnect coupling noise using in-situ delay change curve measurements," in *Proc. Int. Conf. ASIC/SoC*, Sep. 2000, pp. 321–325.

[30] K. Agarwal, T. Sato, Y. Cao, D. Sylvester, and C. Hu, "Efficient generation of

delay change curves for noise-aware static timing analysis," in *Proc. Asia South Pacific Des. Autom. Conf.*, Jan. 2002, pp. 342–348.

[31] D. Das, W. Scott, S. Nazarian, and H. Zhou, "An efficient current-based logic cell model for crosstalk delay analysis," in *Proc. Int. Symp. Quality Electronic Des.*, Mar. 2009, pp. 627–633.

[32] W. Qiu and D. M. H. Walker, "An efficient algorithm for finding the K longest testable paths through each gate in a combinational circuit," in *Proc. Int. Test Conf.*, Sep. 2003, pp. 592–601.

[33] J. Benkoski, E. V. Meersch, L. J. M. Claesen, and H. D. Man, "Timing verification using statically sensitizable paths," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 9, no. 10, pp. 1073–1084, Oct. 1990.

[34] K. J. Keller, H. Takahashi, K. K. Saluja, and Y. Takamatsu, "On reducing the target fault list of crosstalk-induced faults in synchronous sequential circuits," in *Proc. Int. Test Conf.*, Oct. 2001, pp. 568–577.

[35] Z. Wang and D. M. H. Walker, "Dynamic compaction for high quality delay test," in *Proc. VLSI Test Symp.*, May 2008, pp. 243–248.

[36] I. Pomeranz and S. M. Reddy, "On static compaction of test sequences for synchronous sequential circuits," in *Proc. Des. Autom. Conf.*, Jun. 1996, pp. 215–220.

[37] Z. Wang and D. M. H. Walker, "Compact delay test generation with a realistic low

cost fault coverage metric," in *Proc. VLSI Test Symp.*, May 2009, pp. 59–64.

[38] P. S. Zuchowski, P. A. Habitz, J. D. Hayes, and J. H. Oppold, "Process and environmental variation impacts on ASIC timing," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 2004, pp. 336–342.

VITA

Name:	Dibakar Gope
Address:	C/O Dr. Duncan M. (Hank) Walker Department of Computer Science and Engineering Texas A&M University College Station, TX 77843-3112
Email Address:	dibakar@tamu.edu
Education:	B.E., Electrical and Electronics Engineering, Birla Institute of Technology and Science, India, 2008 M.S., Computer Engineering, Texas A&M University, 2011